Spis treści

Podziękowania

Świat wokół nas porusza się na wiele skomplikowanych i pięknych sposobów. Początkowe części naszego życia spędziliśmy, ucząc się naszego środowiska za pomocą percepcji i interakcji. Oczekujemy, że fizyczny świat wokół nas będzie zachowywał się zgodnie z naszą pamięcią percepcyjną, np. jeśli upuścimy kamień, to spadnie on z powodu grawitacji, jeśli wieją porywy wiatru, lżejsze obiekty będą przez niego przenoszone. Ta lekcja skupia się na zrozumieniu, symulacji i dołączaniu elementów naszego świata opartych na ruchu do tworzonych przez nas światów cyfrowych. Mamy nadzieję, że utworzymy intuicyjne, bogate i satysfakcjonujące doświadczenia, korzystając z percepcyjnych wspomnień naszych użytkowników.

– James Tu, opis kursu *Dynamic Bodies*, wiosna 2003, ITP

W 2003 roku, jako dyplomant programu ITP (Interactive Telecommunications Program) w Tisch School of the Arts na New York University, zapisałem się na kurs o nazwie Dynamic Bodies. Kurs ten był prowadzony przez adiunkta w ITP, projektanta interakcji Jamesa Tu. Moja praca skupiała się wtedy na serii eksperymentów programistycznych, które generowały nierealistyczne obrazy w czasie rzeczywistym. Aplikacje obejmowały przechwytywanie obrazów z żywego źródła i „malowanie" barw za pomocą elementów, które poruszały się po ekranie zgodnie z różnymi regułami. Kurs prowadzony przez Tu – obejmujący wektory, siły, oscylacje, układy cząstek, rekurencję, kierowanie i sprężyny – idealnie pasował do mojej pracy.

Wykorzystywałem te pojęcia nieformalnie w swoich własnych projektach, ale nigdy nie poświęciłem czasu, aby dokładnie przeanalizować naukę związaną z tymi algorytmami lub nauczyć się technik obiektowych, aby sformalizować ich implementację. W tym samym semestrze zapisałem się także na kurs prowadzony przez Philipa Galantera w Foundations of Generative Art Systems, który skupiał się na teorii i praktyce sztuki generacyjnej i obejmował takie tematy, jak chaos, automaty komórkowe, algorytmy generyczne, sieci neuronowe i fraktale. Zarówno kurs Tu, jak i Galantera otworzyły mi oczy na świat algorytmów symulacyjnych – technik, które prowadziły mnie w ciągu kilku kolejnych lat pracy i nauczania, służąc za podstawę i inspirację dla tej książki.

Jednak w tej historii brakuje kawałka układanki.

Kurs Galantera był oparty przede wszystkim na teorii, zaś Tu uczył za pomocą Macromedia Director oraz języka programowania Lingo. W tamtym semestrze nauczyłem się wielu algorytmów, tłumacząc je na język C++ (język, którego w tamtym okresie używałem w sposób niezgrabny, a było to, zanim pojawiły się kreatywne środowiska kodowania, takie jak openFrameworks i Cinder). Jednak pod koniec semestru odkryłem coś, co nazywało się Processing (*https://www.processing.org)*. Oprogramowanie to było wtedy w fazie alfa (wersja 0055), a ja miałem trochę doświadczenia z Javą i byłem zaintrygowany na tyle, aby zadać pytanie: Czy ten przyjazny artystom język programowania i środowisko open source mogą być właściwym miejscem do tworzenia zestawu podręczników i przykładów dotyczących programowania i symulacji? Mając wsparcie społeczności ITP i Processing, rozpocząłem coś, co dziś jest niemal dwudziestoletnią podróżą poprzez nauczanie kodowania.

Na początek chciałbym podziękować Redowi Burnsowi, który kierował ITP przez pierwszych 30 lat i zmarł w 2013 roku. Red wspierał i zachęcał mnie do pracy przez ponad 10 lat. Dan O'Sullivan, prodziekan w Emerging Media na uczelni Tisch School of the Arts, był przede wszystkim moim mentorem i pierwszą osobą, która zasugerowała, abym zaczął opracowywać podręczniki do programowania. Shawn Van Every, obecny dziekan wydziału, był moim współpracownikiem podczas mojego pierwszego roku nauczania na pełny etat i stanowił w tym okresie bogate źródło pomocy i inspiracji. Jestem wdzięczny za wsparcie i zachęty ze strony profesor Luisy Pereiry z ITP. Jej praca nad przygotowywaną książką *Code of Music* była wielce inspirująca. Jej innowacyjne podejście do interakcyjnych materiałów edukacyjnych pomogło mi przemyśleć i przedefiniować mój własny proces pisania i publikowania.

Wibrujące i ożywcze środowisko ITP ukształtowało wiele niezwykłych indywidualności. Niezależnie od tego, czy byli to koledzy z pierwszych lat koncepcji tej książki, czy też nowe twarze wnoszące fale inspiracji, jestem wdzięczny pracownikom wydziału ITP/IMA. Na podziękowania zasługują: Ali Santana, Allison Parrish, Blair Simmons, Clay Shirky, Craig Protzel, Danny Rozin, David Rios, Gabe Barcia-Colombo, Katherine Dillon, Marianne Petit, Marina Zurkow, Matt Romein, Mimi Yin, Nancy Hechinger, Pedro Galvao Cesar de Oliveira, Sarah Rothberg, Sharon De La Cruz, Tom Igoe oraz Yeseul Song.

Pełen zaangażowania i niestrudzony personel ITP i IMA (Interactive Media Arts) odegrał ogromną rolę w utrzymaniu ekosystemu w rozkwicie i tworzeniu wszystkiego. Dziękuję wielu osobom, z którymi pracowałem przez wiele lat. Są to: Adrian Mandeville, Brian Kim, Daniel Tsadok, Dante Delgiacco, Edward Gordon, Emma Asumeng, George Agudow, John Duane, Lenin Compres, Luke Bunn, Marlon Evans, Matt Berger, Megan Demarest, Midori Yasuda, Phil Caridi, Rob Ryan, Scott Broussard i Shirley Lin.

Specjalne podziękowanie kieruję do pomocniczych adiunktów wydziału, Ellen Nickles i Nuntinee Tansrisakul, które w 2021 roku nauczały ze mną asynchronicznej wersji online kursu *Nature of Code*, pomimo szczytu światowej pandemii. Ich wkład i idee z tego semestru bardzo wzbogaciły materiały kursu.

Studenci ITP i IMA, a jest ich zbyt wielu, żeby ich wymieniać, stanowili niezwykłe źródło informacji zwrotnych w całym procesie pisania. Wiele materiałów z tej książki pochodzi z mojego kursu pod tym samym tytułem, który prowadziłem 17 razy. Mam stosy roboczych wydruków książki z uwagami pisanymi na marginesach, a także rozległe archiwum e-maili od studentów z poprawkami, komentarzami i hojnymi słowami wsparcia.

Chciałbym wyróżnić kilkoro studentów, którzy pracowali jako dyplomanci nad materiałami do książki. Podczas pracy w projekcie ITP/IMA Equitable Syllabus, Briana Jones i Chaski No zapewnili niezwykłe wsparcie badawcze, które rozwinęło pojęcia i odwołania w tej książce. Jako asystentka w licencjackiej wersji klasy Nature of Code, Gracy Whelihan udzieliła nieocenionego wsparcia i uwag oraz zawsze przypominała mi o pięknie liczb losowych.

Jason Gao i Stuti Mohgaonkar pracowali nad budową systemów tworzenia materiałów książki, wprowadzając nowe przepływy pracy w pisaniu i edycji. Elias Jarzombek także zasługuje na wymienienie za jego rady i wsparcie techniczne, wynikające z projektu książki *Code of Music*.

Po zrobieniu dyplomu Jason Gao kontynuował projektowanie i rozwijanie systemu kompilacji książki i jej witryny internetowej (*https://natureofcode.com)*. Jeśli skierujecie się tam teraz, zobaczycie owoce jego wielu talentów: pełną wersję książki, która gładko integruje się z edytorem sieciowy p5.js. Jego realizacja znacznie wykracza poza moją początkową wizję.

Wnętrze tej książki i witryna internetowa zostały starannie zaprojektowane przez Tuan Huang. Zaczęła ona rozwijać pomysły związane z układem książki wiosną 2003 roku podczas zajęć z Nature of Code. Po dyplomie udoskonaliła projekt, pracując nad stworzeniem spójnego języka wizualnego łączącego różne elementy książki. Jej minimalistyczna i elegancka estetyka znacznie rozwinęła wizualną atrakcyjność i dostępność książki. Specjalne podziękowania należą się OpenMoji (*https://openmoji.org*) – projektowi open source emotikonów i ikon (Licencja Creative Commons CC BY-SA 4.0) – za zapewnienie zachwycającego i szerokiego zbioru emotikonów używanych w różnych miejscach książki.

Mam także dług wdzięczności dla energetycznej i wspierającej społeczności kodowania oraz Processing Foundation. Nie pisałbym tej książki bez Casey Reas i Bena Frya, którzy w 2001 roku stworzyli Processing i współtworzyli Processing Foundation. Poświęcili ponad 20 lat na zbudowanie i utrzymanie oprogramowania i społeczności. Połowę swoje wiedzy zawdzięczam czytaniu kodu źródłowego i dokumentacji

Processing. Elegancka prostota języka Processing, witryny i IDE stanowi oryginalne źródło inspiracji dla całej moje pracy i nauczania.

Lauren Lee McCarthy, twórczyni p5.js zasiała ziarno, które sprawiło, że udało się przekształcić tę książkę na kod JavaScript. Jest niestrudzoną orędowniczką integracji i dostępu do open source, a jej podejście do budowania społeczności było dla mnie ogromnie inspirujące. Cassie Tarakajian stworzyła edytor sieciowy p5.js, heroiczne przedsięwzięcie, które umożliwiło zebranie i uporządkowanie wszystkich przykładów kodu z tej książki.

Moje szczere podziękowania kieruję do obecnych i poprzednich członków zarządu Processing (wraz z Casey, Benem i Laureen): Dorothy Santos, Johanna Hedva, Kate Hollenbach i Xin Xin. Specjalne podziękowanie należy się kierującym projektem, personelu i absolwentów fundacji, którzy odegrali kluczową rolę w kształtowaniu i napędzaniu społeczności i jej projektów. Są to Andres Colubri, Charles Reinhardt, Evelyn Masso, Jesse C. Thompson, Jonathan Feinberg, Moira Turner, Qianqian Ye, Rachel Lim, Raphael de Courville, Saber Khan, Suhyun (Sonia) Choi, Toni Pizza, Tsige Tafesse oraz Xiaowei R. Wang.

W rozdziale 10 wprowadzam projekt m15.js, bibliotekę towarzyszącą p5.js, której celem jest danie kreatywnym programistom możliwości uczenia się maszyn w sposób przyjazny i dostępny. Dziękuję wielu uczonym i studentom z ITP/IMA, którzy dołożyli się do rozwoju. Są to: Apoorva Avadhana, Ashley Lewis, Bomani McClendon, Christina Dacanay, Cristobal Valenzuela, Lydia Jessup, Miaoye Que, Micaelle Lages, Michael Weinberg, Orpheas Kofinakos, Ozi Chukwukeme, Sam Krystal, Yining Shi i Ziyuan (Peter) Lin. Dziękuję profesorowi J.H. Moonowi, profesorowi Gottfriedowi Haiderowi oraz Quinn Fangqing He z NYU Shanghai, który dodatkowo wspierał rozwój biblioteki i był uprzejmy przeczytać pierwsze wersje rozdziałów dotyczących sieci neuronowych. Linda Paiste zasługuje na wzmiankę o jej wysiłkach w celu poprawy bazy kodu. Wreszcie przekazuję specjalne podziękowania Joey K. Lee, który zapewnił cenne wsparcie i uwagi zarówno dla samej książki, jak i dla rozwoju m14.js.

Chciałbym też podziękować badaczowi AI, Davidowi Ha, którego badania dotyczące neuroewolucji (patrz „Additional Resources" na witrynie książki) zainspirowały mnie do stworzenia przykładów implementacji tej techniki za pomocą m15.js i dodania nowego rozdziału do tej książki.

Przez ostatnich 10 lat większość czasu spędziłem na tworzeniu podręczników wideo na moim kanale YouTube, The Coding Train. Jestem niezwykle wdzięczny za ogromne wsparcie i współpracę ze strony bardzo wielu osób, utrzymujących Coding Train w ruchu i na właściwych torach (podczas gdy ja ciężko pracuję nad zmianami kierunku). Są to między innymi: Chloe Desaulles, Cy X, David Snyder, Dusk Virkus, Elizabeth Perez, Jason Heglund, Katie Chan, Kline Gareth, Kobe Liesenborgs oraz Mathieu Blanchette. Specjalne podziękowania kieruję do Melissy Rodriguez, która

pomogła znaleźć i uzyskać pozwolenia na użycie zdjęć rozpoczynających każdy rozdział.

Podziękowania kieruję też do platformy streamingowej Nebula i jej prezesa Dave'a Wiskusa za niezłomne wsparcie, i do twórcy Nebuli, Grady'ego Hillhouse'a, który poradził mi współpracę z wydawnictwem No Strach Press, które wydrukowało ten cholerny tekst. Nie byłbym w stanie dotrzeć do tak szerokiej rzeszy odbiorców bez platformy YouTube, więc specjalne podziękowania kieruję do znakomitego menedżera partnerów YouTube Deana Kowalskiego, a także do Doreen Tran, która pomaga w kierowaniu YouTube Skilling w Ameryce Północnej.

Mam też myślących, mądrych, hojnych i życzliwych widzów. Chciałbym zwłaszcza podziękować następującym osobom: Dipam Sen, Francis Turmel, Kathy McGuinessoraz Simon Tiger, oferujących rady, uwagi, poprawki, wsparcie techniczne i nie tylko to. Dzięki nim książka jest znacznie lepsza.

Chcę też podziękować wielu osobom, które współpracowały ze mną ponad 10 lat temu nad wydaniem z roku 2012. Są to: David Wilson (okładka i projekt książki), Rune Madsen i Steve Klise (budowa systemu i witryny), Shannon Fry (edycja), Evan Emolo, Miguel Bermudez oraz wszyscy wspierający Kickstarter, którzy pomogli sfinansować tę pracę.

Specjalne wyróżnienie należy się Zannah Marsh, która bez wytchnienia pracowała nad stworzeniem ponad 100 ilustracji do wersji książki z 2012 . Jakimś cudem zgodziła się, aby ponownie wykonać tę pracę do nowego wydania. Chcę jej szczególnie podziękować za cierpliwość i chęć podążania za mną, gdy zbyt wiele razy zmieniałem zdanie na temat poszczególnych ilustracji. No i te koty! Uśmiecham się za każdym razem, gdy widzę, jak piszą na klawiaturze.

A teraz prawdziwy powód, dla którego się tu spotykamy. Jestem prawie pewien, że gdyby nie No Starch Press, nie czytalibyście tych słow. Oczywiście moglibyście znaleźć zaktualizowane podręczniki na witrynie internetowej, ale współpraca, wsparcie oraz przemyślane i uprzejme ustalanie terminów przez zespoły naprawdę przepchnęły mnie przez wyboje. Chcę wyrazić ogromną wdzięczność wydawcy Nathanowi Heidelbergerowi, który jest odpowiedzialny za to, że książka ta ma sens, nie mówiąc o uzasadnionych śmiesznych żartach. (Wina za te wszystkie złe leży wyłącznie po mojej stronie). Dziękuję Jasperowi Palfree, redaktorowi technicznemu, który cierpliwie tłumaczył mi tak wiele razy, aż zrozumiałem, jaka jest różnica między ruchem liniowym a obrotowym (i wyjaśnił inne niezliczone pojęcia z zakresu nauki i kodu). Chcę też przekazać specjalne podziękowania redaktorce Sharon Wilkey, której skrupulatna dbałość o szczegóły wygładziły każde zdanie i dodały idealne wykończenie. Ponadto dziękuję Audrey Doyle za bystre oko przy korekcie tekstu. Dziękuję też założycielowi No Starch, Billowi Pollockowi, który nauczył mnie wszystkiego, czego potrzebowałem o zakupach w Trader Joe, redaktor prowadzącej Jill Franklin za jej życzliwość

i cierpliwe wsparcie oraz zespołowi produkcji kierowanemu przez starszą redaktor produkcji Jennifer Kepler i menedżerkę produkcji Sabrinę Plomitallo-Gonzalez, którzy dostosowali się do mojego nietypowego sposobu pracy Notion → GitHub → PDF.

Wreszcie z serca dziękuję mojej żonie Aliki Caloyeras, która ma zawsze rację. Poważnie, ma jakieś super moce. Kocham cię. A także naszym dzieciom – Eliasowi, który łaskawie pozwala mi zachować godność, nie niszcząc mnie całkowicie w koszykówce i grach wideo, oraz Olympii, która przypomina mi, że „czuję się na 22", gdy gramy w tryktraka i w karty i razem się śmiejemy. Chcę też podziękować mojemu ojcu, Bernardowi Shiffmanowi, który szczodrze dzielił się ze mną swoją wiedzą matematyczną i cały czas kierował do mnie uwagi, oraz moje matce, Doris Yaffe Shiffman, i bratu Jonathanowi Shiffmanowi, którzy zawsze bardzo mnie wspierali, zadając pytanie „Jak tam ci idzie praca nad książką?".

Wprowadzenie

Ponad dziesięć lat temu samodzielnie opublikowałem *The Nature of Code*, zasób internetowy i drukowaną książkę dotyczącą badań nieprzewidywalnych ewolucyjnych i pojawiających się właściwości natury za pomocą oprogramowania i kreatywnej struktury kodowania Processing. Stwierdzenie, że od tego czasu wiele się zmieniło w świecie technologii i kreatywnych mediów, to zbyt mało powiedziane, gdyż jest to temat stulecia. Dlatego jestem tutaj ponownie, z nową i zrestartowaną wersją tej książki opartą na JavaScripcie i bibliotece p5.js. Tym razem książka zawiera kilka nowych sztuczek kodowania, ale to ta sama stara natura – ptaki wciąż machają skrzydłami, a jabłka wciąż spadają nam na głowy.

Czym jest ta książka?

Od 2004 roku prowadzę w ITP/IMA (Tisch School of the Arts, New York University) kurs zatytułowany Introduction to Computational Media. Początki tych zajęć sięgają 1987 roku i pracy Mike'a Millsa i Johna Henry'ego Thompsona (wynalazcy języka programowania Lingo). W ramach tego kursu studenci uczą się podstaw programowania (zmienne, instrukcje warunkowe, pętle, obiekty, tablice), a także pojęć związanych z tworzeniem interaktywnych projektów medialnych (piksele, dane, dźwięk, sieci, 3D i inne). W 2008 roku połączyłem moje materiały tego kursu w książce wprowadzającej, *Learning Processing*, a w 2015 roku stworzyłem serię samouczków wideo, które podążają tą samą ścieżką, lecz w JavaScripcie z biblioteką p5.js.

Gdy uczeń pozna już podstawy i zapozna się z szeregiem aplikacji, jego następnym krokiem może być zagłębienie się w konkretny obszar. Może skupić się na wizji komputerowej, wizualizacji danych lub poezji generatywnej. Kolejnym krokiem może też być mój kurs Nature of Code (również prowadzony w ITP/IMA od 2008 roku). Rozpoczyna się on dokładnie tam, gdzie kończy się mój materiał wprowadzający i pokazuje świat technik programowania, które skupiają się na algorytmach i symulacjach. Książka, którą czytacie, wyewoluowała z tego kursu.

Mój cel dla tej książki jest prosty: chcę przyjrzeć się zjawiskom, które naturalnie występują w świecie fizycznym i dowiedzieć się, jak napisać kod w celu ich symulacji.

Czym więc dokładnie jest ta książka? Czy jest to książka o naukach przyrodniczych? Odpowiedź brzmi – zdecydowanie nie. To prawda, mogę analizować tematy, które pochodzą z fizyki lub biologii, ale nie będę ich badał, stosując się do poważnego rygoru akademickiego. Zamiast tego książka jest „inspirowana rzeczywistymi

wydarzeniami". Wybieram fragmenty nauk przyrodniczych i matematyki potrzebne do zbudowania programowej interpretacji natury i zbaczam z kursu lub pomijam szczegóły według własnego uznania.

Czy jest to książka o sztuce lub projektowaniu? Również powiedziałbym, że nie. Niezależnie od tego, jak nieformalne może być moje podejście, wciąż skupiam się na algorytmach i powiązanych z nimi technikach programowania. Oczywiście otrzymane pokazy są wizualne (przejawiają się jako animowane szkice p5.js), ale są to dosłowne wizualizacje samych algorytmów i technik programowania, narysowane wyłącznie za pomocą podstawowych kształtów i kolorów w skali szarości. Mam jednak nadzieję, że wy, drodzy czytelnicy, wykorzystacie swoją kreatywność i pomysły wizualne, aby stworzyć na podstawie tych przykładów nowe, wciągające prace. (Nie będę narzekał, jeśli zmienicie każdy szkic w tęczę).

W końcu, jeśli ta książka jest jeszcze czymś, to jest staromodnym podręcznikiem programowania. Podczas gdy temat z zakresu nauk przyrodniczych (fizyka newtonowska, wzrost komórek, ewolucja) może stanowić zalążek rozdziału, a wyniki mogą inspirować projekty artystyczne, sama treść zawsze sprowadza się do implementacji kodu, ze szczególnym naciskiem na programowanie obiektowe.

Słowo o p5.js

Biblioteka p5.js jest przeformułowaniem kreatywnego środowiska kodowania Processing na potrzeby nowoczesnej sieci. Używam jej w tej książce z kilku powodów. Przede wszystkim jest to środowisko, które jest mi bardzo dobrze znane. Wprawdzie oryginalny Processing zbudowany na Javie jest moją pierwszą miłością i nadal do niego się zwracam, gdy wypróbowuję nowy pomysł, lecz to p5.js używam do nauczania na wielu moich zajęcia z programowania. Jest darmowy, jest open source i dobrze nadaje się dla początkujących, a ponieważ jest to JavaScript, wszystko działa bezpośrednio w przeglądarce internetowej, więc nie wymaga instalacji.

Jednak dla mnie Processing i p5.js to przede wszystkim społeczność ludzi, a nie biblioteki czy ramy postepowania. Ci ludzie poświęcili niezliczone godziny na tworzenie i udostępnianie oprogramowania. Napisałem tę książkę dla tej społeczności i dla każdego, kto uwielbia zaspakajać swoją dociekliwość i bawić się kodem.

Podsumowując, nic nie wiąże treści tej książki ściśle z p5.js ani z Processing. Ta książka mogłaby zostać napisana przy użyciu zwykłego JavaScriptu lub Javy albo dowolnej liczby innych kreatywnych środowisk kodowania typu open source, takich jak openFrameworks, Cinder i tak dalej. Mam nadzieję, że po ukończeniu tej książki będę w stanie wydać wersje przykładów działające w innych środowiskach. Jeśli ktoś jest zainteresowany pomocą w przeniesieniu tych przykładów, proszę o kontakt pod adresem *daniel@natureofcode.com*. Śmiało, jeśli wiesz, że chcesz przenieść *Naturę kodu* do PHP!

Wszystkie przykłady w tej książce zostały przetestowane za pomocą p5.js w wersji 1.9.0, ale w większości powinny działać również z wcześniejszymi wersjami. Będę je aktualizować do najnowszej wersji. Najnowszy kod można zawsze znaleźć na stronie internetowej tej książki (*https://natureofcode.com*) i w powiązanym z nią repozytorium GitHub (*https://github.com/nature-of-code)*.

Co trzeba wiedzieć?

Wymagania wstępne do zrozumienia materiału zawartego w tej książce można określić jako „jeden semestr nauki programowania w p5.js, Processing lub innym kreatywnym środowisku kodowania". Nie ma więc przeszkód, aby czytać tę książkę, ucząc się programowania w innym języku lub środowisku programistycznym.

Jeśli nigdy wcześniej nie pisaliście żadnego kodu, to możecie przeczytać tę książkę, aby poznać koncepcje i inspiracje, ale prawdopodobnie będziecie mieć trudności z kodem, ponieważ zakładam znajomość podstaw: zmiennych, instrukcji warunkowych, pętli, funkcji, obiektów i tablic. Jeśli te pojęcia są dla was nowe, to wszelkie podstawy zawierają moje kursy *Code! Programming with p5.js* (*https://thecodingtrain.com/p5js*) i *Learning Processing* (*https://thecodingtrain.com/processing)*.

Doświadczeni programiści, którzy nie pracowali jeszcze z p5.js, prawdopodobnie mogą to szybko nadrobić, sięgając po dokumentację p5.js (*https://p5js.org)*, przeglądając przykłady i czytając stronę biblioteki *Get Started* (*https://p5js.org/get-started)*.

Trzeba również zaznaczyć, że dość istotne jest doświadczenie w programowaniu obiektowym. W rozdziale 10 omawiam niektóre z podstaw tej tematyki, ale jeśli klasy i obiekty nie są wam znane, sugeruję obejrzenie moich samouczków wideo dotyczących programowania obiektowego p5.js i Processing, dostępnych również w Coding Train (*https://thecodingtrain.com/oop)*.

Jak czytasz tę książkę?

Czy czytasz tę książkę na Kindle? Na papierze? Na laptopie w formacie PDF? Na tablecie wyświetlającym animowaną wersję HTML5? A może jesteś przywiązany do krzesła, wchłaniając treść bezpośrednio do mózgu za pomocą szeregu elektrod, rurek i kaset?

Moim marzeniem zawsze było napisanie tej książki w jednym formacie (w tym przypadku w zbiorze dokumentów Notion), a następnie, po naciśnięciu magicznego przycisku (`npm run build`), wychodzi gotowa książka w dowolnym formacie – PDF, HTML5, wydrukowana na papierze, dla Kindle i tak dalej. Stało się to w dużej mierze możliwe dzięki projektowi Magic Book (*https://github.com/magicbookproject)*, który jest strukturą open source do samodzielnego publikowania treści, pierwotnie

opracowaną w ITP przez Rune Madsena i Steve'a Klise'a. Wszystko zostało zaprojektowane i wystylizowane przy użyciu CSS – bez ręcznego składu i makiety.

Rzeczywistość, w której powstała ta książka, nie jest tak czysta, a historia tego, jak do tego doszło, jest długa. Jeśli chcecie dowiedzieć się więcej, koniecznie przeczytajcie zawarte tu podziękowania, a następnie zatrudnijcie osoby, którym podziękowałem, aby pomogli wydać waszą książkę! Więcej szczegółów zamieszczam też w powiązanym repozytorium GitHub (*https://github.com/nature-of-code)*.

Najważniejszy jest fakt, że materiał jest taki sam bez względu na format, w jakim go czytasz. Jedyną różnicą będzie sposób korzystania z przykładów kodu – więcej na ten temat w punkcie „Jak czytać kod" na stronie xxxiii.

Połączenie z Coding Train

Osobiście nadal uwielbiam połączenie masy celulozowej, starannie złączonej odpornym grzbietem, na którym zostały artystycznie rozmieszczone barwne składniki, aby przekazać słowa i idee. Jednak od 2012 roku, kiedy pod wpływem impulsu nagrałem w moim biurze w ITP swoją pierwszą lekcję wideo na temat programowania, odkryłem ogromną wartość i radość z przekazywania pomysłów i lekcji za pomocą ruchomych obrazków.

Krótko mówiąc, mam kanał YouTube o nazwie *The Coding Train* (*https://www.youtube.com/thecodingtrain)*. Wspomniałem o nim wcześniej, omawiając możliwości zapoznania się ze wstępnymi materiałami do tej książki, a jeśli będziecie dalej czytać, zobaczycie, że w całej książce odwołuję się do związanych z tym filmów. Mogę nawiązać do jednego z moich filmów na temat powiązanego algorytmu lub alternatywnej techniki dla konkretnego przykładu kodowania, lub zasugerować serię na temat pokrewnego pojęcia, co może wprowadzić dodatkowy kontekst dla analizowanego przeze mnie tematu.

Jeśli lubicie uczyć się za pomocą wideo, pracuję również nad dodatkowym zestawem samouczków wideo, które zawierają dokładnie ten sam materiał, co ta książka. Dziesięć lat temu utworzyłem ich całe mnóstwo przy użyciu Processing, a ostatnio zacząłem publikować serię aktualizacji przy użyciu p5.js. W chwili pisania tego tekstu jestem mniej więcej w połowie rozdziału 5.

Dodatkowe materiały

Istnieje również mnóstwo wyjątkowych materiałów edukacyjnych uczących symulacji i algorytmów generatywnych, których nie zapisałem ani nie nagrałem. Zawsze zalecam, aby podczas nauki nowych tematów zapoznawać się z różnymi perspektywami i głosami. Możliwe, że to, co napisałem, nie trafi do was, podobnie jak może nie pomóc słuchanie, gdy powtarzam te same informacje w formie wideo, niezależnie od

tego jak bardzo będę się starał przed kamerą. Czasami lepiej jest, gdy ktoś inny pisze, mówi lub pokazuje te same koncepcje innymi słowami i w innym stylu. W tym celu na stronie internetowej książki zamieszczam część „Dodatkowe materiały". Jeśli tworzycie własne materiały lub macie jakieś zalecenia dotyczące ich włączenia, skontaktujcie się ze mną!

Dwie szybkie rekomendacje, które mam pod ręką, to *The Computational Beauty of Nature* autorstwa Gary'ego Williama Flake'a (MIT Press, 1998) – to stamtąd pierwotnie zaczerpnąłem wiele pomysłów na tę książkę – oraz doskonale zorganizowane źródło online *That Creative Code Page* autorstwa Taru Muhonena i Raphaela de Courville'a (*https://thatcreativecode.page)*.

Opowieść o tej książce

Patrząc na spis treści książki, można zobaczyć 12 rozdziałów (0-11!), z których każdy obejmuje inny temat. I w pewnym sensie ta książka jest właśnie tym – przeglądem kilkunastu koncepcji i powiązanych z nimi przykładów kodu. Niemniej jednak, układając materiał, zawsze wyobrażałem sobie coś w rodzaju liniowej narracji. Zanim zaczniecie lekturę, chciałbym oprowadzić was po tej opowieści.

Część 1: Przedmioty nieożywione

Piłka do footballu leży na trawie. Kopnięcie wyrzuca ją w powietrze. Grawitacja ściąga ją z powrotem na dół. Silny podmuch wiatru utrzymuje ją w powietrzu jeszcze przez chwilę, aż spadnie i odbije się od głowy skaczącego zawodnika.

Piłka nie jest żywa – nie dokonuje żadnych wyborów co do tego, jak będzie się poruszać po świecie. Jest raczej nieożywionym obiektem czekającym na popychanie i ciągnięcie przez siły otoczenia.

Jak zamodelować piłkę nożną poruszającą się na cyfrowym płótnie? Jeśli kiedykolwiek programowaliście okrąg poruszający się po ekranie, to zapewne napisaliście następujący wiersz kodu:

```
x = x + 1;
```

Rysujemy kształt w położeniu `x`. Z każdą kolejną ramką animacji zwiększamy wartość `x`, przerysowujemy kształt i gotowe – iluzja ruchu! Może też pójść o krok lub dwa dalej i dołączyć położenie `y`, a także zmienne dla prędkości wzdłuż osi x i y:

```
x = x + xspeed;
y = y + yspeed;
```

Część 1 tej opowieści rozwinie ten pomysł jeszcze bardziej. Po zbadaniu sposobu używania różnych rodzajów losowości do napędzania ruchu obiektu (**rozdział 0**), zamierzam wziąć te zmienne `xspeed` i `yspeed` i pokazać, jak razem tworzą one wektor (**rozdział 1**). Nie powstaną z tego żadne nowe funkcjonalności, ale zostaną zbudowane solidne podstawy programowania ruchu na potrzeby pozostałej części książki.

Gdy dowiecie się co nieco o wektorach, szybko zdacie sobie sprawę, że siła (**rozdział 2**) jest wektorem. Kopnięcie futbolówki to przyłożenie siły. Co siła robi z obiektem? Według Sir Isaaca Newtona siła jest równa masie pomnożonej przez przyspieszenie, więc siła powoduje przyspieszenie obiektu. Modelowanie sił pozwala tworzyć systemy z dynamicznym ruchem, w których obiekty poruszają się zgodnie z różnorodnymi zasadami.

Piłka, do której przyłożono siłę, może się również obracać. Obiekt poruszający się zgodnie ze swoim przyspieszeniem *liniowym*, może obracać się ze swoim przyspieszeniem *kątowym* (**rozdział 3**). Zrozumienie podstaw kątów i trygonometrii pozwala modelować obracające się obiekty, a także zrozumieć zasady ruchu oscylacyjnego, jak kołysanie się wahadła lub odchylanie się sprężyny.

Gdy już uporacie się z podstawami ruchu i sił dla pojedynczego obiektu nieożywionego, pokażę wam, jak utworzyć tysiące takich obiektów i zarządzać nimi jak pojedynczą jednostką zwaną *układem cząstek* (**rozdział 4**). Układy cząstek są również dobrym pretekstem do przyjrzenia się dodatkowym cechom programowania obiektowego – dziedziczeniu i polimorfizmowi.

Część 2: To żyje!

Co to znaczy modelować życie? Nie jest łatwo odpowiedzieć na to pytanie, ale zacznę od budowania obiektów, które mają zdolność postrzegania swojego otoczenia. Zastanówmy się nad tym przez chwilę. Klocek, który spada ze stołu, porusza się zgodnie z siłami, podobnie jak delfin pływający w wodzie. Jest jednak kluczowa różnica: klocek nie może zdecydować się na skok ze stołu, podczas gdy delfin może zdecydować się na wyskoczenie z wody. Delfin ma marzenia i pragnienia. Odczuwa głód i strach, a te odczucia wpływają na jego ruchy. Analizując techniki modelowania autonomicznych agentów (**rozdział 5**), nauczycie się, jak tchnąć życie w nieożywione obiekty, pozwalając im podejmować decyzje dotyczące ich ruchów zgodnie z tym, jak rozumieją one otoczenie.

Wszystkie przykłady z rozdziałów od 1 do 5 zostały napisane „od zera", co oznacza, że kod algorytmów kierowania ruchem obiektów został utworzony bezpośrednio w p5.js. Z pewnością nie jestem jednak pierwszym programistą, który rozważał pomysł symulowania fizyki i życia w animacji, więc w następnej kolejności badam, w jaki sposób można wykorzystać biblioteki fizyki (**rozdział 6**) do modelowania

bardziej wyrafinowanego zachowania. Omawiam funkcje dwóch bibliotek: Matter.js i Toxiclibs.js.

Pod koniec rozdziału 5 analizuję zachowania grupowe, które wykazują cechy systemu złożonego. Taki system jest zwykle definiowany jako system, który jest czymś więcej niż tylko sumą jego części. Podczas gdy poszczególne elementy systemu mogą być niezwykle proste i łatwe do zrozumienia, zachowanie systemu jako całości może być bardzo złożone, inteligentne i trudne do przewidzenia. Pogoń za złożonością prowadzi od myślenia wyłącznie o modelowaniu ruchu do sfery systemów opartych na regułach. Co można modelować za pomocą automatów komórkowych (**rozdział 7**), systemów komórek żyjących na siatce? Jakie rodzaje wzorów można wygenerować za pomocą fraktali (**rozdział 8**), geometrii natury?

Część 3: Inteligencja

Sprawiliśmy, że rzeczy się poruszyły. Następnie daliśmy tym rzeczom nadzieje, marzenia i obawy, wraz z zasadami, według których mają żyć. Ostatnim krokiem w tej książce będzie wprowadzenie do naszych dzieł inteligentnego podejmowania decyzji. Czy do systemów obliczeniowych można zastosować biologiczny proces ewolucji (**rozdział 9**) w celu ewolucji zachowania autonomicznych agentów? Czy można zaprogramować sztuczną sieć neuronową (**rozdział 10**), czerpiąc inspirację z ludzkiego mózgu? W jaki sposób agenty mogą podejmować decyzje, uczyć się na błędach i dostosowywać się do otoczenia (**rozdział 11**)?

Korzystanie z tej książki jako sylabusa

Mimo że treść tej książki z pewnością wymaga intensywnego i mocno skompresowanego semestru, zaprojektowałem ją tak, aby pasowała do 14-tygodniowego kursu. Zauważyłem, że niektóre rozdziały sprawdzają się, gdy są rozłożone na dwa lub więcej tygodni, podczas gdy inne można połączyć i analizować razem w jednym tygodniu. Oto jeden z możliwych programów nauczania:

Tydzień 1	Losowość i wektory (rozdziały 0–1)
Tydzień 2	Siły (rozdział 2)
Tydzień 3	Oscylacja (rozdział 3)
Tydzień 4	Układy cząstek (rozdział 4)
Tydzień 5	Autonomiczne agenty (rozdział 5)
Tydzień 6	Biblioteki fizyki (rozdział 6)
Tydzień 7	Projekt w połowie semestru dotyczący ruchu
Tydzień 8	Systemy złożone: dwuwymiarowe komórkowe i fraktale (rozdziały 7–8)

Tydzień 9	Algorytmy genetyczne (rozdział 9)
Tydzień 10	Sieci neuronowe i neuroewolucja (rozdziały 10–11)
Tydzień 11	Omówienie tematu projektu końcowego
Tygodnie 12–13	Warsztaty dotyczące projektu końcowego
Tydzień 14	Prezentacja projektu końcowego

Jeśli rozważacie wykorzystanie tego tekstu na kursie lub podczas warsztatów, skontaktujcie się ze mną. Mam nadzieję, że w końcu uda mi się ukończyć towarzyszący mu zestaw filmów, a także dołączyć pomocne slajdy jako dodatkowe materiały edukacyjne. Jeśli stworzycie własne, chętnie o tym usłyszę!

Jak czytać kod

Kod jest w tej książce głównym środkiem przekazu i przeplata się przez całą narrację, gdy jest rozkładany na części i analizowany. Czasami pojawia się jako pełne, samodzielne przykłady, innym razem jako pojedynczy wiersz lub dwa, a często rozciąga się na całe podrozdziały w wielu krótkich fragmentach, z objaśnieniami umieszczonymi pomiędzy nimi. Niezależnie od formy, kod zawsze będzie wyświetlany `czcionką stałopozycyjną`. Oto krótki przewodnik po tym, jak poruszać się po typach kodu rozsianych po całej książce.

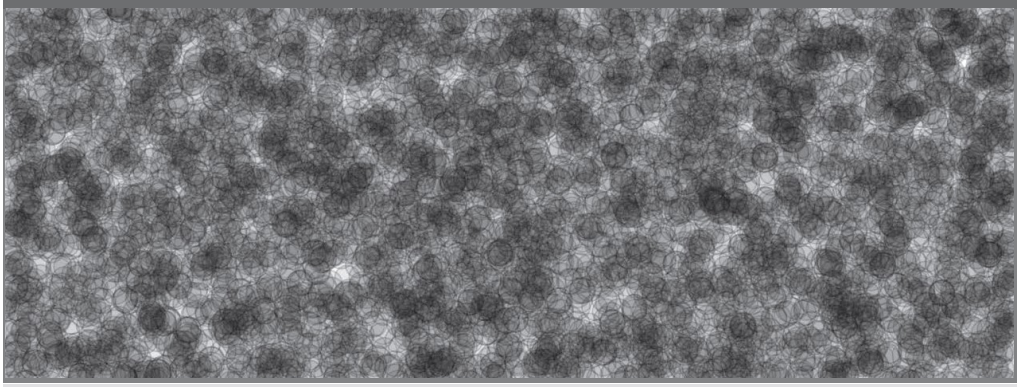
Pełne przykłady

Każdy rozdział zawiera w pełni funkcjonalne przykłady kodu napisane przy użyciu biblioteki p5.js. Jak one wyglądają, można zobaczyć na sąsiedniej stronie.

W każdym rozdziale przykłady są kolejno ponumerowane, aby ułatwić znalezienie odpowiedniego kodu online. W drukowanej wersji książki zrzut ekranu znajduje się tuż pod tytułem przykładu. W wersji online szkic jest osadzony bezpośrednio na stronie. W przypadku animowanych przykładów (a prawie wszystkie są takie), zrzuty ekranu często pokazują „ścieżkę" ruchu. Efekt ten został osiągnięty poprzez dodanie przezroczystości do funkcji `background(255, 10)`, choć załączony kod nie zawiera tego ulepszenia.

Poniżej przykładu znajduje się kod, ale nie zawsze jest to *pełny* kod. Ponieważ wiele przykładów jest dość długich i obejmuje wiele plików, dokładam wszelkich starań, aby dołączyć fragment, który podkreśla główne aspekty przykładu lub jakiekolwiek nowe elementy, które zostały w danej części wprowadzone, a nie zostały jeszcze omówione wcześniej.

Przykład #.# *Tytuł przykładu*



```
function setup() {
  createCanvas(640, 240);                    // Ten rozmiar płótna* jest używany w celu dostosowania układu książki, ale nie jest istotny dla przykładów.
  background(255);
}

function draw() {
  fill(0, 25);
  stroke(0, 50);
  circle(random(width), random(height), 16);  // Za każdym razem narysuj losowy okrąg za pomocą funkcji draw().
}
```

Pełną wersję kodu można znaleźć na stronie internetowej książki. Można tam wchodzić w interakcje, modyfikować i eksperymentować z kodem w edytorze p5.js Web Editor (*https://editor.p5js.org*). Ponadto wszystko jest zawarte w repozytorium GitHub książki. Poniżej znajdują się linki do wszystkich materiałów:

- Strona internetowa książki (*https://natureofcode.com*) zawiera pełny tekst książki (w języku angielskim), dodatkowe lektury i materiały oraz wszystkie przykłady kodu.

* Czytelnika może początkowo dziwić słowo „płótno" używane w kontekście obrazów komputerowych. Wystarczy jednak pomyśleć o naszych algorytmach i programach jak o malarzach. Powierzchnia, na której powstaje obraz, zwyczajowo nazywa się płótnem, bo ten właśnie materiał – odpowiednio przygotowany („zagruntowany") i naciągnięty na drewnianą ramę – od wieków służy jako podłoże dla obrazów olejnych (choć oczywiście nie tylko ten). Niekiedy mianem „płótna" określa się także także gotowe arcydzieło. Zatem „płótno" (ang. *canvas*) to po prostu ta przestrzeń ekranu, w której będą pojawiać się obrazy generowane przez nasz kod (wszystkie przypisy pochodzą od tłumacza lub redaktora wydania polskiego).

- Repozytoria GitHub (*https://github.com/nature-of-code*) zawierają surowy kod źródłowy strony internetowej książki, proces kompilacji książki i wszystkie przykłady kodu.
- Oprócz strony internetowej i repozytoriów GitHub, można również uzyskać dostęp do kodu, przeglądając listę szkiców w edytorze internetowym p5.js (*https://editor.p5js.org/natureofcode/sketches)*.

Warto zauważyć, że w przykładzie użyłem komentarzy, aby odnieść się do tego, co dzieje się w kodzie. Komentarze te znajdują się obok kodu (ich wygląd może się różnić w zależności od sposobu czytania książki). Cieniowanie tła grupuje komentarze z odpowiadającymi im wierszami kodu.

Pełne fragmenty

Chociaż rzadko, „pełne" części kodu są czasami mieszane z tekstem głównym. Czasami, tak jak w przypadku przykładu #.# w poprzednim punkcie, mogę zaprezentować cały kod powiązany z pełnym szkicem p5.js. W większości przypadków jednak za „pełny" fragment kodu uznaję kod całej funkcji lub klasy – w pełni ukończony blok kodu, wraz z otwierającymi i zamykającymi nawiasami klamrowymi i wszystkim, co znajduje się między nimi. Coś w tym rodzaju:

Cała funkcja draw() dla przykładu

```
function draw() {
  background(255);
  for (let x = 0; x < width; x += spacing) {
    fill(255);
    circle(x, height / 2, spacing);
  }
}
```

Ten fragment pokazuje całą funkcję `draw()`, ale nadal nie jest to kompletny szkic. Zakłada on istnienie zmiennej globalnej o nazwie `spacing`, a także funkcji `setup()`, która wywołuje funkcję `createCanvas()`.

Kod bez kontekstu

Od czasu do czasu na stronie można znaleźć wiersze kodu bez otaczającej je funkcji lub kontekstu. Te fragmenty mają na celu zilustrowanie danej kwestii, ale niekoniecznie mają być w danej postaci uruchamiane. Mogą one przedstawiać pojęcie, niewielki fragment algorytmu lub technikę kodowania.

```
  fill(240, 99, 164);                    Wartości RGB, aby okręgi były
                                         różowe
```

Warto zauważyć, że ten bezkontekstowy fragment pasuje do wcięcia wiersza `fill(255)` w poprzednim „pełnym" fragmencie. Robię to, gdy kod stanowi część czegoś pokazanego wcześniej. Co prawda, nie zawsze będzie to działać tak czysto lub idealnie, ale staram się jak mogę!

Wycięty kod

Uważaj na nożyczki! Ten element projektu wskazuje, że fragment kodu jest kontynuacją poprzedniego fragmentu lub będzie kontynuowany po pewnym tekście wyjaśniającym. Czasami nie jest on naprawdę kontynuowany, ale jest po prostu odcinany, ponieważ cały kod nie jest istotny dla danej dyskusji. Nożyczki są po to, by powiedzieć: „Hej, może być więcej tego kodu powyżej lub poniżej, ale w każdym razie jest to część czegoś większego!". Oto, jak może to wyglądać z otaczającym tekstem.

Pierwszym krokiem do stworzenia szkicu p5.js jest utworzenie płótna:

```
function setup() {
  createCanvas(640, 240);
```

Następnie nadszedł czas na narysowanie tła:

```
  background(255);
```

Lubię też umieszczać okrąg na środku płótna:

```
  circle(width / 2, height / 2, 200);
}
```

W funkcji `draw()` mogę zacząć umieszczać kwadraty w losowych miejscach tła i w ustalonym okręgu. Reszta kodu może być dowolna!

```
function draw() {
  rectMode(CENTER);
  square(random(width), random(height), 20);
```

Warto zauważyć, że zachowuję spójne wcięcia, aby pomóc w ustaleniu kontekstu i znów używam ikony nożyczek, aby wskazać, gdzie kod jest kontynuowany lub ucięty.

Szczególnym efektem ubocznym korzystania z fragmentów kodu jest fakt, że często można zauważyć otwierające nawiasy klamrowe w jednym fragmencie, które mają odpowiedni nawias zamykający dopiero kilka fragmentów kodu później (jeśli w ogóle). Jeśli jesteście przyzwyczajeni do czytania kodu w JavaScripcie, może to początkowo wywołać lekką panikę, ale miejmy nadzieję, że się do tego przyzwyczaicie.

Ćwiczenia

Każdy rozdział zawiera ponumerowane ćwiczenia, które służą jako przestrzeń do zabawy, eksperymentów i wykraczania poza pojęcia i kod przedstawiony w rozdziałach. Oto jak może wyglądać ćwiczenie:

>  **Ćwiczenie #.#**
>
> Spróbuj zmodyfikować przykład #.# tak, aby każdy okrąg miał losowy rozmiar:
>
> ```
> function draw() {
>   fill(0, 25);
>   stroke(0, 50);
>   circle(random(width), random(height),                );
> }
> ```

Abyście byli czujni, ćwiczenia są dostępne w różnych formatach. Niektóre z nich stanowią wyzwania techniczne, wymagając napisania odmiany konkretnego algorytmu lub rozwiązania ściśle określonego problemu. Inne mają charakter otwarty, zachęcając do zabawy i eksperymentowania i podążania za własnymi pomysłami. Niektóre zawierają fragmenty kodu z pustymi miejscami, które zachęcają do ich bezpośredniego wypełnienia. Nie wahajcie się pisać lub bazgrać w tej książce, podczas ich przerabiania!

Rozwiązania

Rozwiązania ćwiczeń znajdują się na stronie internetowej książki. Powinienem raczej powiedzieć, że dążę do umieszczenia rozwiązań wszystkich ćwiczeń na stronie internetowej książki. W tej chwili dostępnych jest tylko kilka z nich, ale mam nadzieję, że do czasu, gdy to przeczytasz, będzie ich znacznie więcej. Jeśli chcielibyście dodać rozwiązanie do któregoś z ćwiczeń, z przyjemnością zrobię to za pośrednictwem repozytorium GitHub książki!

Projekt ekosystemu

Choć chciałbym udawać, że wszystkiego można się nauczyć, wylegując się w wygodnym fotelu i czytając prozę, to aby nauczyć się programowania, trzeba naprawdę trochę programować. Ćwiczenia rozrzucone po każdym rozdziale są dobrym początkiem, ale pomocne może okazać się również pamiętanie o bardziej istotnym pomyśle na projekt (lub dwa), który można rozwijać, przechodząc od rozdziału do rozdziału. W istocie, prowadząc w ITP mój kurs Nature of Code, często odkrywałem, że studenci lubią w tracie semestru tworzyć pojedynczy projekt, krok po kroku, tydzień po tygodniu.

Na końcu każdego rozdziału znajduje się seria podpowiedzi dla jednego takiego projektu – ćwiczenia, które opierają się na sobie nawzajem, po jednym temacie. Ten projekt jest oparty na następującym scenariuszu. Zostaliście poproszeni przez muzeum nauki o opracowanie oprogramowania dla nowej wystawy, *Cyfrowy ekosystem*, świata animowanych, proceduralnych stworzeń, które żyją w komputerowej symulacji, aby cieszyć swym działaniem odwiedzających muzeum. Nie mam zamiaru sugerować, że jest to szczególnie innowacyjna lub kreatywna koncepcja. Wykorzystuję raczej ten przykładowy projekt ekosystemu jako dosłowną reprezentację treści zawartych w książce, pokazując, w jaki sposób elementy mogą pasować do siebie w jednym programie. Zachęcam do opracowania własnego pomysłu, być może bardziej abstrakcyjnego i nietradycyjnego.

Uzyskiwanie pomocy i przesyłanie opinii

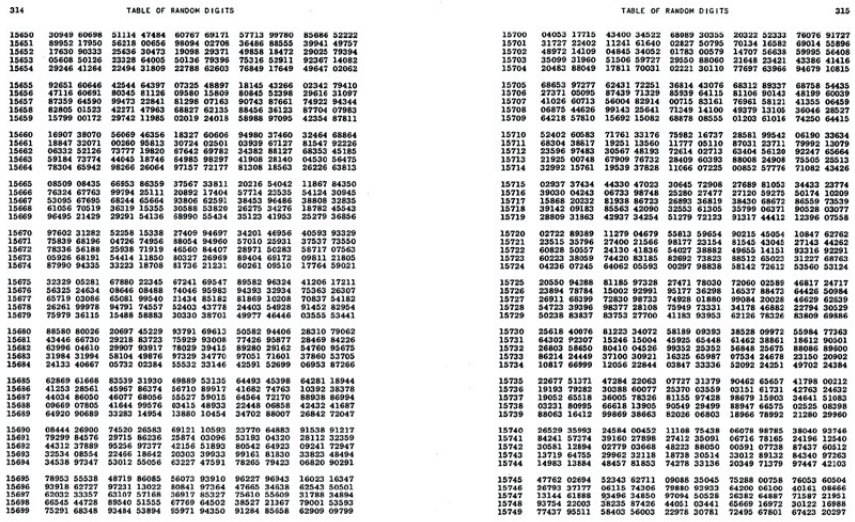
Kodowanie może być trudne i frustrujące, a pomysły zawarte w tej książce nie zawsze są proste. Nie musicie robić tego sami. Prawdopodobnie czyta to właśnie teraz ktoś, kto chciałby współorganizować grupę badawczą lub klub książki, gdzie można się spotkać, porozmawiać i podzielić się swoimi zmaganiami i sukcesami. Jeśli nie znajdziecie lokalnej społeczności, z którą można wspólnie odbyć tę podróż, to może zorganizować społeczność internetową? Dwa miejsca, które sugeruję, to oficjalne fora Processing (*https://discourse.processing.org*) i serwer Discord Coding Train (*https://thecodingtrain.com/discord)*.

Uważam, że wersja online tej książki jest żywym dokumentem i czekam na waszą opinię. Wszystkie informacje związane z książką można znaleźć na stronie Nature of Code (*https://natureofcode.com)*. Surowy tekst źródłowy książki i wszystkie ilustracje znajdują się w GitHubie (*https://github.com/nature-of-code)*. Prosimy o pozostawienie opinii i przesłanie poprawek za pomocą zgłoszeń GitHuba (*https://github.com/nature-of-code/noc-book-2/issues)*.

Co ważniejsze, chcę zobaczyć, co tworzycie! Możecie dzielić się swoimi pomysłami, przesyłając je do kolekcji pasażerów pociągu (Passenger Showcase) na stronie Coding Train (*https://thecodingtrain.com/showcase*) lub w kanałach na wspomnianym Discordzie. Komentarze na YouTube są zawsze mile widziane (choć szczerze mówiąc, często lepiej ich nie czytać) i nie krępujcie się oznaczać mnie na dowolnej platformie, jaką mają do zaoferowania w przyszłości media społecznościowe – którakolwiek jest najbardziej przyjazna i najmniej toksyczna! Chcę cieszyć się wszystkimi bloopami, które pływają w waszych ekosystemach. Niezależnie od tego, czy triumfalnie przeskakują przez falę kreatywności, czy też robią mały plusk w stawie nauki, wygrzewajmy się w falach, które wysyłają poprzez naturę kodowania!

0 Losowość

Generowanie liczb losowych jest zbyt ważne,
aby pozostawić to przypadkowi.
– Robert R. Coveyou



314 TABLE OF RANDOM DIGITS

15650	30949	60698	51114	47484	60767	69171	57713	99780	85686 52222
15651	89952	17950	56218	00656	98094	02708	36486	88555	39941 49757
15652	17630	90332	25626	30473	19098	29371	49858	18472	29025 79394
15653	05608	50126	23328	64005	50136	79396	75316	52911	92367 14082
15654	29246	41264	22494	31809	22788	62603	76849	17649	49647 02062
15655	92651	60646	42544	64397	07325	48897	18145	43266	02342 79410
15656	47116	60691	80345	81126	09580	15809	80845	52398	29616 31097
15657	87359	64590	99473	22841	81296	07163	90743	87661	74922 94344
15658	82805	01523	42271	47963	68827	62135	88456	36122	87704 07983
15659	15799	00172	29742	11985	02019	24018	58988	97095	42354 87811
15660	16907	38070	56069	46356	18327	60606	94980	37460	32464 68864
15661	18847	32071	00260	95813	30724	02501	03939	67127	81547 92226
15662	06333	52126	72777	19820	67642	69782	34382	88127	68353 45185
15663	59184	73774	44045	18746	64985	98297	41908	28140	04530 56475
15664	78304	65942	98266	26064	97157	72177	81308	18563	26226 63813
15665	08509	08435	66953	86359	37567	33811	20216	54042	11867 84350
15666	76334	67763	99784	25111	20892	17404	57714	23535	54124 30945
15667	53095	67695	58244	65664	93806	62591	38453	96486	38808 32835
15668	61056	70519	36119	15355	30588	53820	26275	34276	18782 45543
15669	96495	21429	29291	54136	68990	55434	35123	41953	25279 36856
15670	97602	31282	52258	15338	27409	94697	34201	46956	40593 93329
15671	75839	68196	04726	74956	88054	94960	57010	28931	37537 73550
15672	78336	56188	25938	71919	46560	84407	28971	56283	58717 07563
15673	05926	68191	54414	11850	80327	26969	89404	69172	09811 21805
15674	87990	94335	33223	18708	81736	21231	60261	08510	17764 39021
15675	32329	05281	67880	22345	67241	69547	89582	96324	41206 17211
15676	56325	24634	08646	08488	74046	95982	94293	32934	75363 26307
15677	65719	03086	65081	99540	21434	85182	81869	10208	70837 54182
15678	26261	99978	94791	74557	52403	43778	24403	54828	91452 82954
15679	75979	36115	15488	58883	30330	38701	49977	46446	03555 53441
15680	88580	80026	20697	45229	93791	69613	50582	94406	28310 79062
15681	43446	66730	29218	83723	75829	93008	77426	95877	28469 84226
15682	63996	04610	29907	93917	78029	39415	89280	29162	54760 95475
15683	31984	31994	58104	49876	97329	34770	97051	71601	37860 53705
15684	24133	40667	05732	02384	55532	33146	42591	52699	06953 87266
15685	62869	61668	83539	31930	69889	53135	64493	45398	64281 18944
15686	41253	28561	45967	86374	56710	89917	41682	74763	10392 38378
15687	44034	86050	46077	88056	55527	59015	64564	72170	88938 86994
15688	09669	07805	41644	99576	03415	48933	22449	06858	42432 41687
15689	64920	90689	33283	14954	13880	10454	24702	88007	26842 72047
15690	08444	26900	74520	26583	69121	10593	23770	64883	91538 91217
15691	79299	84576	29715	86236	25874	03096	53193	04320	28112 32359
15692	44312	37889	95206	97277	42156	51893	80042	64923	09241 72947
15693	32534	08554	22466	18642	20303	39933	99161	81830	33823 48494
15694	34538	97347	53012	55056	63227	47591	78265	79423	06820 90291
15695	78953	55538	48719	86085	56073	93910	96227	96343	16023 16347
15696	93916	62727	97231	13022	80841	97364	47665	34638	63543 50501
15697	63032	33357	63107	57168	36917	85327	75610	55609	31788 34894
15698	66545	44728	88540	51555	67769	64502	38527	21367	79001 53593
15699	75291	68348	93484	53894	85971	94350	91284	85658	62909 09799

TABLE OF RANDOM DIGITS 315

15700	04053	17715	43400	34522	68989	30355	20322	52333	78076 91727
15701	31727	22402	11241	61640	02827	50795	70134	16582	69014 55896
15702	48972	14109	04845	34052	01783	00579	14707	56638	59995 56408
15703	35099	31960	51506	59727	29550	88060	21648	23421	43386 41416
15704	20483	88049	17811	70031	02221	30110	77697	63966	94679 10815
15705	68653	97277	62431	72251	36814	43076	68312	89337	68758 54435
15706	27371	05095	87439	71329	85939	64115	81106	90143	48199 60039
15707	41026	60713	56004	82914	00715	83161	76961	58121	41355 06459
15708	06875	44626	99143	25641	71549	14100	49379	13106	39046 28527
15709	64218	57810	15692	15082	68878	08555	01203	61016	74250 64415
15710	52402	60583	71761	33176	75982	16737	28581	99542	06190 32634
15711	68304	38617	19251	13560	11777	05110	87031	23711	79992 13079
15712	23596	97483	30567	48193	72614	02713	63404	56129	92247 65664
15713	21925	00748	67909	76732	28409	60393	88008	24908	78505 28513
15714	32992	15761	19539	37828	11066	07225	00852	57776	71082 43426
15715	02937	37434	44330	47022	30645	72908	27689	81053	34433 22774
15716	39030	04243	06732	98748	25380	27477	27120	59275	50174 10209
15717	15868	20232	81938	86723	26893	36819	38430	68672	86559 73539
15718	39142	09183	85563	42090	32553	61305	35799	06271	90528 03077
15719	28809	31863	42937	34254	51279	72123	91317	44412	12396 07558
15720	02722	89389	11279	04679	55813	59654	90215	45054	10847 62762
15721	23515	35796	27400	21566	98177	23154	81545	43045	27143 44262
15722	60828	50557	24130	41836	54027	38882	49653	14151	93316 92291
15723	60223	38059	74420	83185	82692	73823	88512	65023	31227 68763
15724	04236	07245	64062	05593	00297	98838	58142	72611	53560 53124
15725	20550	94388	81185	97328	27471	78030	72060	02585	46817 24717
15726	23894	78784	15002	92991	95177	36298	16537	88473	64426 50984
15727	26911	68399	72830	98733	74928	01880	99084	20028	46629 62639
15728	54723	29296	98377	28108	75949	73321	34178	46882	22794 30529
15729	50238	83837	83753	27700	41183	93953	62126	76326	83809 69886
15730	25618	40076	81223	34072	58189	09393	38528	09972	55984 77363
15731	64302	92307	15246	15004	45925	45448	61462	38861	18612 90501
15732	26803	58450	80410	04526	99352	25352	56848	25675	88086 89900
15733	86214	24449	37100	30921	16325	65987	07534	24678	23150 20902
15734	10817	66999	12056	22844	03847	33336	52092	24251	49702 24384
15735	22677	51371	47284	22063	07727	31379	90462	65657	41798 00212
15736	19183	79282	30388	60077	25370	03559	02151	61731	42763 24632
15737	19052	65518	36005	78326	81155	97428	98679	15903	34641 51083
15738	02231	80955	66618	13905	90549	29499	88947	66575	02525 08398
15739	88063	16412	99869	38663	82026	06803	18966	78992	21280 29960
15740	26529	35993	24584	00452	11108	75438	06078	98785	38040 93746
15741	84241	57274	39160	27898	27412	35091	06716	78165	24196 12540
15742	30581	12894	02779	03668	48233	88050	00591	07738	87437 60112
15743	13719	64755	29962	32118	18738	30514	33012	89132	84340 97263
15744	14983	13884	48457	81853	74278	33136	20349	71379	97447 42103
15745	47762	02694	52343	62711	09088	35045	75288	00758	76053 60504
15746	36793	37177	06115	74366	79880	93933	64200	06100	40161 08666
15747	13144	61888	93496	34850	97094	50528	26382	64887	71587 21951
15748	93754	22003	28235	87426	44051	03441	65669	16972	30122 18988
15749	77437	95511	58403	56003	22978	30781	72495	67801	67423 20297

Tablice liczb losowych z książki *A Million Random Digits with 100 000 Normal Deviates*, opublikowane przez RAND Corporation

W 1947 roku RAND Corporation opublikowała dziwaczną książkę, pod podanym wyżej tytułem. Nie było to dzieło literackie ani traktat filozoficzny o losowości. Zawierała tablicę liczb losowych wygenerowanych przy użyciu symulacji elektronicznej za pomocą koła ruletki. Książka ta była jedną z ostatnich w serii tabel liczb losowych tworzonych od połowy lat dwudziestych do lat pięćdziesiątych XX wieku. Wraz z rozwojem szybkich komputerów generowanie liczb pseudolosowych stało się szybsze niż odczytywanie ich z tablic i w ten sposób era drukowanych tabel liczb losowych ostatecznie się zakończyła.

Oto jesteśmy na początku. Jeśli minęło trochę czasu, odkąd programujecie w języku JavaScript (lub zajmowaliście się matematyką), ten rozdział przypomni wam myślenie obliczeniowe. Aby zacząć podróż w dziedzinie kodowania natury, podam kilka podstawowych narzędzi do programowania symulacji: liczby losowe, rozkłady losowe i szum. Spójrzcie na to jak na pierwszy (zerowy) element tablicy, która tworzy tę książkę – odświeżenie i brama do możliwości leżących przed wami.

W rozdziale 1 opisuję pojęcie wektora i to, jak służy on jako element konstrukcji symulowania ruchu w całej tej książce. Ale zanim uczynię ten krok, pomyślmy przez chwilę, co oznacza, że coś porusza się na cyfrowym płótnie. Zacznę do najbardziej znanej i najprostszej symulacji ruchu: błądzenia losowego.

Błądzenie losowe

Wyobraź sobie, że stoisz na środku belki równoważni. Co 10 sekund rzucasz monetą. Orzeł – robisz krok do przodu. Reszka – krok do tyłu. To jest właśnie **błądzenie losowe**, ścieżka zdefiniowana jako ciąg losowych kroków. Zeskakując (ostrożnie) z równoważni na podłogę, rozszerzasz błądzenie losowe z jednego wymiaru (poruszania się tylko do przodu i do tyłu) na dwa wymiary (poruszanie się do przodu, do

tyłu, w lewo i w prawo). Ponieważ są teraz cztery możliwości, trzeba dwukrotnie rzucić tą samą monetą, aby określić kolejny krok.

Rzut 1	Rzut 2	Wynik
Orzeł	Orzeł	Krok do przodu
Orzeł	Reszka	Krok w prawo
Reszka	Orzeł	Krok w lewo
Reszka	Reszka	Krok do tyłu

Może się to wydawać dość prostym algorytmem, ale błądzenie losowe może być wykorzystane do modelowania różnego rodzaju zjawisk, które zachodzą w prawdziwym świecie, od ruchów molekuł gazu po karmienie zwierząt i zachowania hazardzisty spędzającego cały dzień w kasynie. Dla naszych celów błądzenie losowe jest z trzech poniższych powodów idealnym początkiem:

- Chciałbym ocenić pomysł programowania stanowiący podstawę tej książki: programowanie obiektowe (OOP, ang. *object-oriented programming*). Błądzenie losowe, które mam zamiar utworzyć, posłuży jako szablon do wykorzystania projektu obiektowego sprawiającego, że elementy będą poruszać się po płótnie grafiki komputerowej.
- Losowe błądzenie wywołuje dwa pytania, które będę wciąż zadawał w tej książce: „Jak zdefiniować reguły, które rządzą zachowaniem naszych obiektów", a następnie „Jak zaimplementować te reguły w kodzie?".
- Zapewne co jakiś czas w ramach prezentowanych w książce projektów będziecie musieli wykazać podstawowe zrozumienie losowości, prawdopodobieństwa i szumu Perlina. Błądzenie losowe pozwoli mi na pokazanie kluczowych elementów, które przydadzą się później.

Najpierw opiszę nieco programowanie obiektowe, kodując klasę `Walker`, aby tworzyć obiekty `Walker` zdolne do losowego błądzenia. Omówienie to będzie dość pobieżne. Osoby, które wcześniej nigdy nie zajmowały się programowaniem obiektowym, mogą potrzebować nieco bardziej wyczerpującego wprowadzenia. Sugeruję w takim przypadku przerwę na zapoznanie się z fragmentem „Objects" mojego kursu wideo *Code! Programming with p5.js* dostępnego na witrynie *https://thecodingtrain.com/objects*.

Klasa losowych obiektów Walker

W języku JavaScript **obiekt** to jednostka, która zawiera w sobie dane i funkcjonalność. W tym przypadku obiekt `Walker` powinien zawierać dane o swoim położeniu na płótnie i funkcjonalność w postaci zdolności do narysowania siebie lub wykonania kroku.

Klasa stanowi szablon do tworzenia konkretnych instancji obiektów. Można traktować klasę jak wycinarkę do ciastek, a wtedy obiekty są tymi ciastkami. Aby utworzyć obiekt `Walker`, zacznę od zdefiniowania klasy `Walker` – określania, co znaczy bycie spacerowiczem*.

Spacerowicz musi mieć tylko dwa elementy danych: liczbę określającą jego położenie na osi x oraz liczbę określającą jego położenie na osi y. Nadam im wartości początkowe na środku płótna, ustalając w ten sposób położenie początkowe obiektu. Mogę to zrobić za pomocą funkcji **konstruktora** klasy, nazwanej odpowiednio `constructor()`. Konstruktora można traktować jako funkcję inicjującą obiekt – `setup()`. Jest ona odpowiedzialna za zdefiniowanie początkowych cech obiektu, podobnie jak to robi funkcja `setup()` dla całego szkicu:

```
class Walker {
  constructor() {                                  Obiekty mają konstruktor, który je
                                                   inicjuje
    this.x = width / 2;
    this.y = height / 2;                           Obiekty mają dane
}
```

Zwróćmy uwagę na słowo kluczowe `this` w celu dołączenia cech do samego nowo utworzonego obiektu: `this.x` oraz `this.y`.

Poza danymi, klasy mogą być zdefiniowane z funkcjonalnością. W tym przykładzie obiekt `Walker` ma dwie funkcje znane w kontekście programowania obiektowego jako **metody**. W istocie metody są funkcjami, ale różnica polega na tym, że metody są zdefiniowane wewnątrz klasy, więc są połączone z obiektem lub klasą, co nie dotyczy funkcji. Słowo kluczowe `function` stanowi wygodną wskazówkę: pojawi się podczas definiowania odrębnych funkcji, ale nie będzie występować w klasie. Staram się używać w tej książce spójnej terminologii, ale programiści często używają wymiennie określeń *funkcja* i *metoda*.

Pierwsza metoda, `show()`, zawiera kod rysowania obiektu (w postaci czarnej kropki). Podczas odwoływania się do cech (zmiennych) danego obiektu nigdy nie zapominajcie o `this`:

* *Walker* to po polsku spacerowicz.

```
show() {                                                    Obiekty mają metody
  stroke(0);
  point(this.x, this.y);
}
```

Kolejna metoda, `step()`, kieruje obiekt `Walker` do następnego kroku. Tu sprawa zaczyna być bardziej interesująca. Pamiętacie wykonywanie kroków po podłodze w losowych kierunkach? Teraz do reprezentowania podłogi wykorzystam płótno p5.js. Istnieją cztery możliwe kroki. Krok w prawo można zasymulować, zwiększając `x` jako `x++`, krok w lewo – zmniejszając `x` w postaci `x--`, krok do przodu, idąc o piksel w górę (`y--`), a krok wstecz – idąc o piksel w dół (`y++`). Ale jak kod będzie wybierał jedną z tych czterech możliwości?

Wcześniej stwierdziłem, że możemy wykonać dwa rzuty monetą. Jednak w p4.js, gdy chcemy losowo wybrać jedną z opcji na liście, możemy po prostu wygenerować liczbę losową, korzystając z funkcji `random()`. Wybiera ona losową wartość zmiennoprzecinkową (dziesiętną) w pożądanym przez nas zakresie. Tu korzystam z wartości 4, aby wskazać zakres od 0 do 4:

```
let choice = floor(random(4));
```

Konwencje kodowania

W języku JavaScript zmienne mogą być deklarowane za pomocą `let` lub `const`. Typowym podejściem jest zadeklarowanie wszystkich zmiennych za pomocą `const` i w razie potrzeby ich zmianę za pomocą `let`. W pierwszym przykładzie odpowiednie do zadeklarowania wyboru będzie `const`, gdyż nigdy nie otrzymuje nowej wartości podczas życia wewnątrz każdego wywołania `step()`. To rozróżnienie jest ważne, ale ja biorę przykład konwencji z p5.js i deklaruję wszystkie zmienne za pomocą `let`.

Rozumiem, że JavaScript z ważnych powodów ma i `let`, i `const`. Jednak rozróżnienie ich może rozpraszać i być mylące dla osób początkujących. Zachęcam do dalszej analizy tego tematu i do podjęcia samodzielnej decyzji, która z metod deklarowania zmiennych jest najlepsza w waszych szkicach. Ponadto można poczytać więcej na ten temat w zagadnieniu #3877 w repozytorium GitHub dla p5.js (*https://github.com/processing/p5.js/issues/3877)*.

Wybieram także korzystanie z operatora ścisłej równości (`===`) w języku JavaScript (i jego odpowiednika w postaci nierówności, `!==`). Ten operator boolowski testuje równość zarówno wartości, jak i typu. Na przykład `3 === '3'` da

w wyniku `false`, ponieważ porównywane są tu dwa różne typy (liczba z łańcuchem), co wymaga przekształcenia, choć wyglądają podobnie. Z drugiej strony użycie operatora luźnej równości `(==)` dla `3 == '3'` da wynik `true`, gdyż typy zostaną przekształcone tak, aby się dały porównać. Wprawdzie luźne porównanie często działa dobrze, ale czasami może prowadzić do nieprzewidywalnych wyników i dlatego `===` jest zapewne bezpieczniejszym wyborem.

Deklaruję zmienną `choice` i przypisuję jej liczbę losową (całkowitą) za pomocą `floor()`, aby usunąć pozycje dziesiętne z losowej liczby zmiennoprzecinkowej. Z technicznego punktu widzenia liczba generowana przez `random(4)` leży w zakresie od zera (włącznie) do 4 (wyłączając), co oznacza, że nigdy nie może osiągnąć wartości `4,0`. Największa możliwa liczba, którą można wygenerować, leży poniżej 4 – to 3,999999999 (z taką liczbą dziewiątek, na jaką pozwala JavaScript). Liczbę tę funkcja `floor()` obcina do 3, usuwając część dziesiętną. Tak więc przypisuję zmiennej `choice` jedną z liczb 0, 1, 2 lub 4.

Następnie `Walker` robi odpowiedni krok (lewo, prawo, góra lub dół), zależnie od tego, jaka liczba losowa zostanie wybrana. Oto cała metoda `step()`, kończąca klasę `Walker`:

```
  step() {
    let choice = floor(random(4));          0, 1, 2 lub 3. Wybór losowy określa krok
    if (choice === 0) {
      this.x++;
    } else if (choice === 1) {
      this.x--;
    } else if (choice === 2) {
      this.y++;
    } else {
      this.y--;
    }
  }
}
```

Gdy już napisaliśmy klasę, nadeszła pora, aby w samym szkicu umieścić konkretny obiekt `Walker`. Zakładając, że szukacie modelu dla pojedynczego losowego błądzenia, należy zacząć od jednej zmiennej:

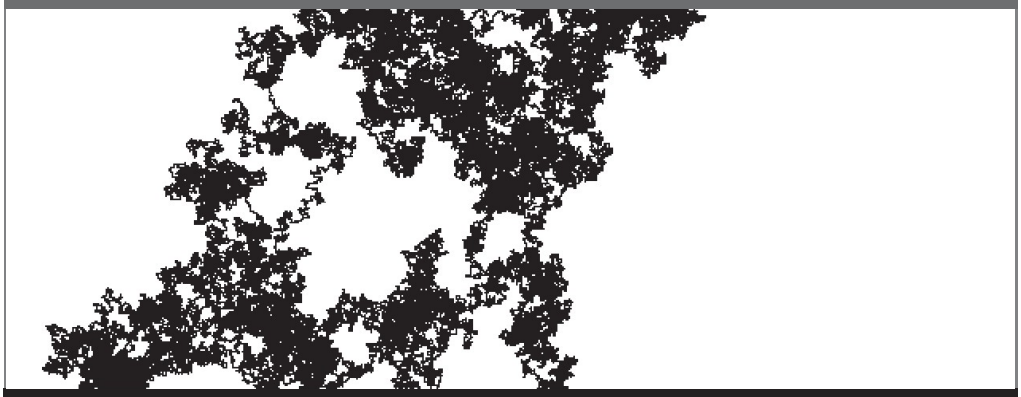
```
let walker;                                 Obiekt Walker
```

Następnie w `setup()` tworzymy obiekt, odwołując się do nazwy klasy za pomocą operatora `new`:

```
function setup() {                                  Pamiętacie, jak działa p5.js? setup() jest wykonywane raz, gdy szkic się rozpoczyna
  createCanvas(640, 240);
  walker = new Walker();                            Utwórz obiekt Walker
  background(255);
}
```

Podczas każdego cyklu `draw()` obiekt `Walker` robi krok i rysuje kropkę.

Przykład 0.1 *Tradycyjne błądzenie losowe*



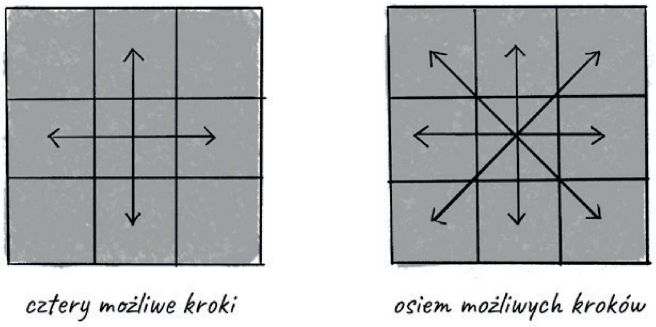
Za każdym razem, gdy pojawiają się te ramki z przykładami, kod do nich jest dostępny w edytorze sieciowym p5.js i na witrynie książki. Osoby czytające książkę offline zobaczą jedynie zrzut ekranu na wynikowym płótnie.

```
function draw() {                                   Kod draw() działa w nieskończonej pętli aż do jej przerwania

  walker.step();                                    Wywołaj funkcję względem obiektu walker

  walker.show();
}
```

Ponieważ tło jest rysowane raz podczas działania `setup()`, a nie jest za każdym razem czyszczone za pomocą `draw()`, ścieżka losowego błądzenia jest widoczna na płótnie.

W błądzeniu losowym można wprowadzić kilka poprawek. Przede wszystkim kroki obiektu `Walker` są ograniczone do czterech opcji: w lewo, w prawo, w górę lub w dół. Ale każdy piksel na płótnie może mieć ośmiu możliwych sąsiadów, w tym na

przekątnych (patrz rysunek 0.1). Dziewiąta możliwość to pozostanie w tym samym miejscu, co także może być opcją do wyboru.



Rysunek 0.1 *Kroki przy błądzeniu losowym z przekątnymi i bez nich*

Aby zaimplementować obiekt `Walker`, który może zrobić krok do dowolnego sąsiedniego piksela (lub pozostać na miejscu), można wybrać liczbę od 0 do 8 (dziewięć możliwych wyborów). Jednak innym sposobem napisania kodu jest wybór spośród trzech możliwych kroków wzdłuż osi x (–1, 0 lub 1) oraz trzech wzdłuż osi y:

```
step() {
  let xstep = floor(random(3)) - 1;          Daje wartość –1, 0 lub 1
  let ystep = floor(random(3)) - 1;
  Yields -1, 0, or 1
  this.x += xstep;
  this.y += ystep;
}
```

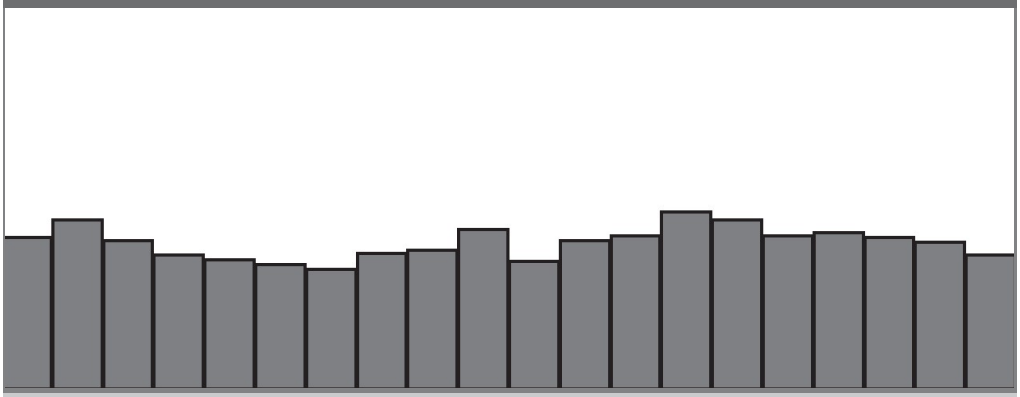
Idąc dalej, można się pozbyć `floor()` i wykorzystać oryginalne liczby zmiennoprzecinkowe z `random()`, aby utworzyć ciągły zakres możliwych długości kroków od –1 do 1, jak to pokazano poniżej.

```
step() {
  let xstep = random(-1, 1);                 Dowolna liczba zmiennoprzecinkowa od –1 do 1
  let ystep = random(-1, 1);
  from -1 to 1
  this.x += xstep;
  this.y += ystep;
}
```

Wszystkie te warianty związane z tradycyjnym losowym błądzeniem mają jeden element wspólny: w dowolnym momencie prawdopodobieństwo, że obiekt `Walker` wykona krok w danym kierunku jest równe prawdopodobieństwu wykonania kroku w każdym innym kierunku. Innymi słowy, jeśli mamy cztery możliwości, to jest jedna szansa na 4 (czyli 25 procent) wykonania kroku w jednym z kierunków. Przy dziewięciu możliwościach jest jedna szansa na 9 (około 11,1 procent).

Tak właśnie działa funkcja `random()`, co jest dla nas wygodne. Generator liczb losowych w p5.js (który działa w tle) tworzy **rozkład normalny** liczb. Można przetestować ten rozkład, licząc, ile razy wybrana zostanie liczba i pokazując te wartości na wykresie.

Przykład 0.2 *Rozkład liczb losowych*



```
let randomCounts = [];                                  Tablica do przechowywania częstości wyboru liczb losowych

let total = 20;                                         Całkowita liczba zliczeń

function setup() {
  createCanvas(640, 240);
  for (let i = 0; i < total; i++) {
    randomCounts[i] = 0;
  }
}

function draw() {
  background(255);
  let index = floor(random(randomCounts.length));       Wybierz liczbę losową i zwiększ licznik
  randomCounts[index]++;
  stroke(0);
  fill(127);
```

```
  let w = width / randomCounts.length;
                                                    Wykreśl wyniki
  for (let x = 0; x < randomCounts.length; x++) {
    rect(x * w, height - randomCounts[x], w - 1, randomCounts[x]);
  }
}
```

Zauważmy, że każdy słupek na wykresie ma nieco inną wysokość. Wielkość próbki (liczba wybieranych liczb losowych) jest mała, więc pojawiają się pewne rozbieżności, gdy jedne liczby są wybierane częściej niż inne. Wraz z upływem czasu dobry generator liczb losowych wyrówna ten rozkład.

Liczby pseudolosowe

Liczby losowe z ciągu `random()` nie są naprawdę losowe. W istocie są **pseudolosowe**, gdyż stanowią wynik funkcji matematycznej, która po prostu symuluje losowość. Funkcja ta da z upływem czasu pewien wzorzec i przestanie wyglądać na losową. Jednak ten czas jest długi, więc funkcja `random()` jest losowa w wystarczającym stopniu na potrzeby przykładów z tej książki.

Ćwiczenie 0.1

Utwórz obiekt błądzący losowo, który ma większą tendencję do poruszania się w dół i w prawo. (Rozwiązanie znajduje się w następnym podrozdziale.)

Prawdopodobieństwo i rozkłady niejednostajne

Losowość jednorodna nie zawsze jest najrozsądniejszym rozwiązaniem problemu – w szczególności w sytuacjach obejmujących tworzenie symulacji organicznych lub wyglądających naturalnie. Jednak za pomocą kilku sztuczek funkcja `random()` może tworzyć **rozkłady niejednostajne** liczb losowych, w których pewne wyniki są bardziej prawdopodobne od innych. Ten rodzaj rozkładu może dać ciekawe wyniki, bardziej przypominające naturalne.

Przypomnijcie sobie, jak zaczęliście programowanie za pomocą p5.js. Zapewne chcieliście narysować na ekranie wiele okręgów, więc powiedzieliście sobie „Już wiem! Narysuję te okręgi na losowych pozycjach, nadając im losowe wielkości i kolory”. Wprowadzanie losowości do systemu jest bardzo rozsądnym punktem wyjścia