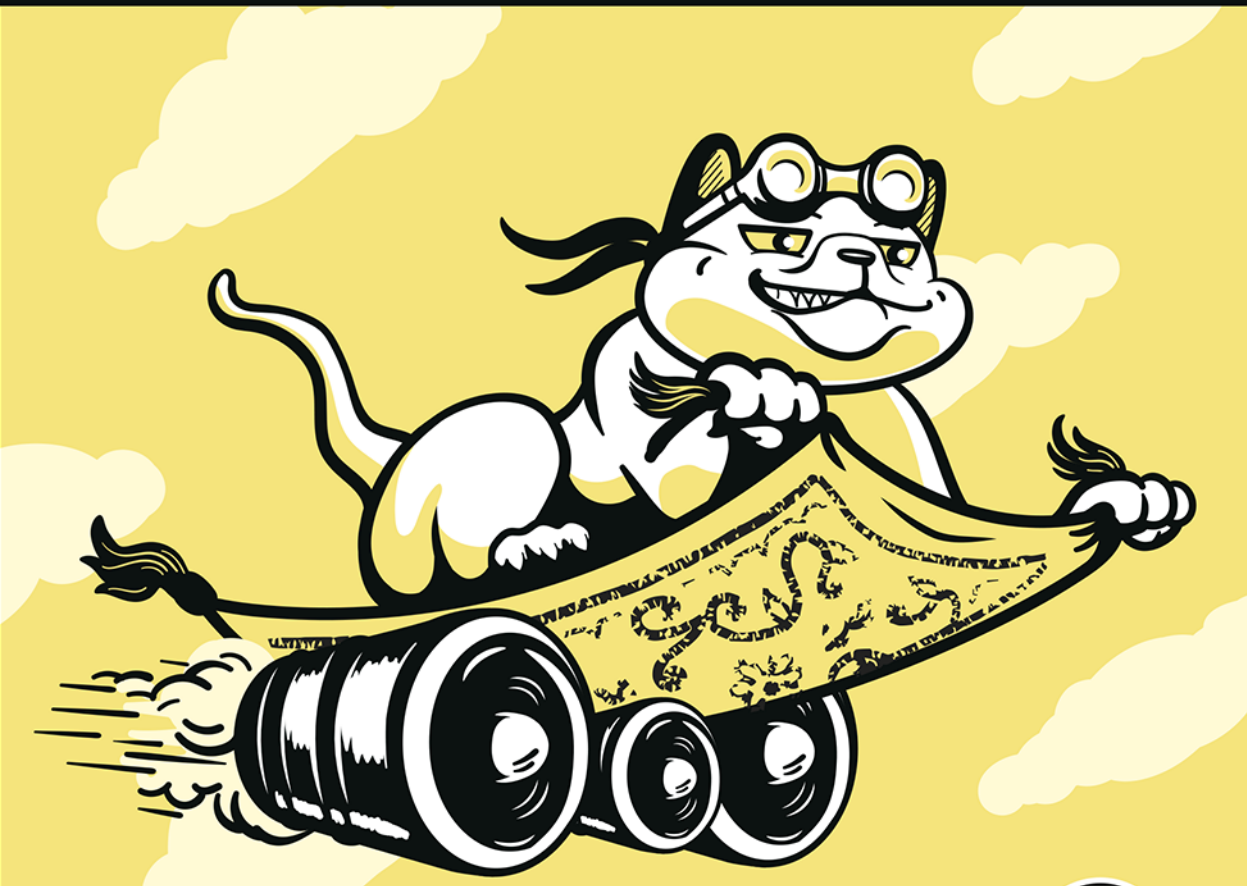
BŁYSKAWICZNY KURS JAVASCRIPT

PRAKTYCZNE WPROWADZENIE DO PROGRAMOWANIA

NICK MORGAN



Helion

no starch press

Tytuł oryginału: JavaScript Crash Course: A Hands-On, Project-Based Introduction to Programming

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-2067-5

Title of English-language original: *JavaScript Crash Course: A Hands-On, Project-Based Introduction to Programming*, ISBN 9781718502260, published by No Starch Press Inc.
245 8th Street, San Francisco, California United States 94103.

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

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

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

Printed in Poland.

Spis treści

10

Pong

Po prezentacji podstaw języka oraz wybranych potężnych technik związanych z używaniem JavaScriptu na stronach internetowych, w celu tworzenia interaktywnych aplikacji, w trzeciej części książki zostanie pokazane, jak zastosować je w praktyce podczas budowania rzeczywistych projektów.

Gra

Gra Pong została opracowana w 1972 roku i wydana w tym samym roku jako niezwykle popularny automat do gier. To jest bardzo prosta gra, w której znajdują się piłeczka oraz dwie paletki ułożone po lewej i prawej stronie ekranu. Tymi paletkami gracze mogą poruszać w górę i w dół. Jeżeli piłeczka uderzy w górną lub dolną krawędź ekranu, odbije się od niej. Natomiast jeśli przekroczy prawą lub lewą krawędź ekranu, wówczas gracz po przeciwnej stronie zdobywa punkt. Piłeczka normalnie odbija się od paletki, chyba że uderzy się ją niemalże górną lub dolną krawędzią paletki — wówczas zmienia się kąt toru piłeczki.

W rozdziale zajmiemy się utworzeniem naszej własnej wersji gry Pong, którą nazwiemy Tennjs (podobnie jak w słowie *tenis*, ale z dodatkiem *JS*). W budowanej grze lewa paletka

będzie kontrolowana przez komputer, natomiast prawa przez gracza. W oryginalnej wersji gry obie paletki były kontrolowane za pomocą kontrolerów obrotowych, w naszej wersji sterowanie będzie odbywało się za pomocą myszy. Komputer zamiast próbować przewidzieć miejsce odbicia piłeczki, zawsze będzie próbował dopasować położenie paletki do położenia piłeczki w pionie. Aby graczowi dać szansę w grze, zdefiniujemy górną granicę szybkości, z jaką komputer może poruszać paletką.

Przygotowania

Rozpoczynamy od zdefiniowania struktury plików projektu oraz utworzenia elementu `canvas`, na którym będzie prowadzona rozgrywka. Jak zwykle projekt wymaga dokumentu HTML i pliku JavaScript. Na początek przygotujemy plik HTML. Utwórz katalog o nazwie *tennjs*, a następnie w nim plik o nazwie *index.html*. Zawartość tego pliku przedstawiłem na listingu 10.1.

Listing 10.1. Plik index.html dla budowanej tutaj gry

```
<!DOCTYPE html>
<html>
  <head>
    <title>Tennjs</title>
  </head>
  <body>
    <canvas id="canvas" width="300" height="300"></canvas>
    <script src="script.js"></script>
  </body>
</html>
```

Mamy do czynienia praktycznie z takim samym plikiem HTML, jaki został utworzony w poprzednim rozdziale. Zatem tutaj nie ma żadnych niespodzianek. Element `body` zawiera element `canvas`, w którym będzie odbywała się gra, a także element `script` odwołujący się do pliku *script.js*, w którym zdefiniujemy kod gry.

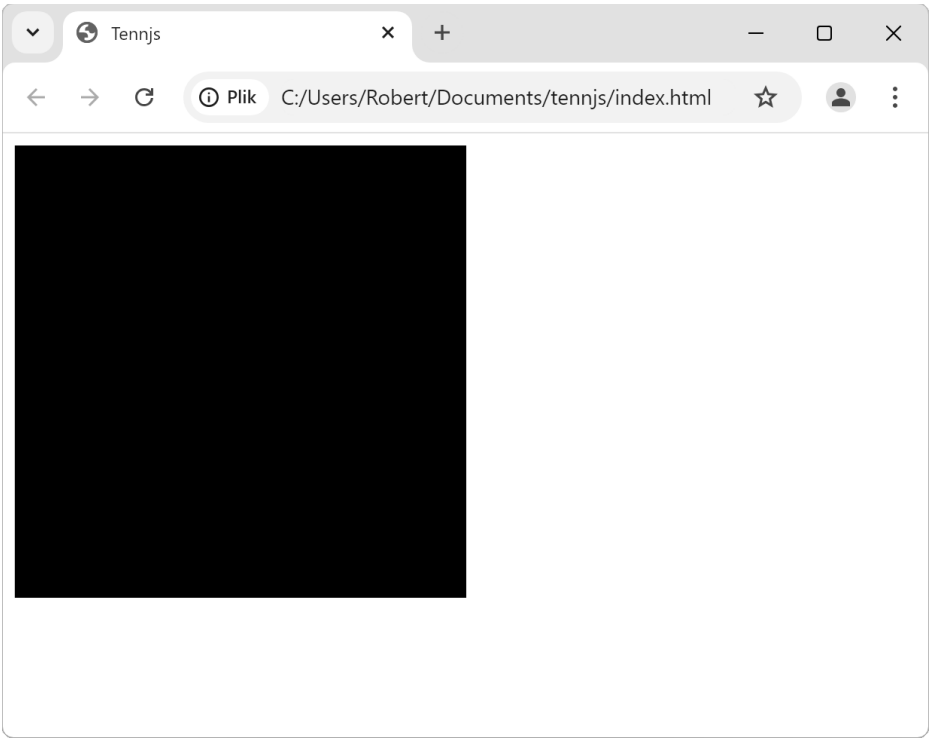
Następnym krokiem jest utworzenie kodu JavaScript przeznaczonego do konfiguracji płótna. Utwórz więc wspomniany wcześniej plik *script.js* i umieść w nim kod z listingu 10.2.

Listing 10.2. Konfiguracja płótna w pliku script.js

```
let canvas = document.querySelector("#canvas");
let ctx = canvas.getContext("2d");
let width = canvas.width;
let height = canvas.height;

ctx.fillStyle = "black";
ctx.fillRect(0, 0, width, height);
```

Ten kod również nie powinien być zaskoczeniem. Na początku za pomocą metody `document.querySelector` znajduje się odwołanie do elementu `canvas` w celu pobrania kontekstu rysowania na płótnie. Dalej długość i wysokość płótna zostają zapisane w zmiennych o nazwach `width` i `height`, aby z poziomu kodu mieć łatwy dostęp do tych wartości. Kolejny krok to zdefiniowanie czarnego koloru wypełnienia i narysowanie czarnego kwadratu o wielkości odpowiadającej wielkości płótna. W ten sposób wydawać się będzie, że płótno ma czarny kolor tła.

Otwórz plik *index.html* w przeglądarce internetowej, a zobaczysz efekt pokazany na rysunku 10.1.



Rysunek 10.1. Nasz czarny kwadrat

W ten sposób otrzymaliśmy puste czarne płótno i możemy przystąpić do utworzenia gry.

Piłeczka

Teraz zajmiemy się narysowaniem piłeczki. Na końcu pliku *script.js* umieść kod przedstawiony na listingu 10.3.

Listing 10.3. Kod odpowiedzialny za narysowanie piłeczki

```
--cięcie--
ctx.fillStyle = "black";
ctx.fillRect(0, 0, width, height);

const BALL_SIZE = 5;
let ballPosition = { x: 20, y: 30 };  ❶

ctx.fillStyle = "white";
ctx.fillRect(ballPosition.x, ballPosition.y, BALL_SIZE, BALL_SIZE);
```

W tym fragmencie kodu metoda `fillRect` została użyta do narysowania piłeczki w postaci małego białego kwadratu w pobliżu lewego górnego rogu płótna. Podobnie jak w pierwotnej wersji gry Pong, piłeczka jest kwadratowa, a nie okrągła. To pozwala zachować klimat retro, a także ułatwia wykrywanie, czy piłeczka zderzyła się ze ścianą bądź paletką. Wielkość piłeczki jest przechowywana w zmiennej o nazwie `BALL_SIZE`. Użyliśmy stylu „prawdziwej stałej", czyli nazwy zapisanej wielkimi literami, ponieważ wielkość piłeczki nie zmieni się w trakcie gry. Wprawdzie można by użyć po prostu wartości `5` zamiast stałej `BALL_SIZE` podczas wywoływania metody `fillRect`, ale w kodzie jeszcze wielokrotnie będziemy odwoływać się do tej wartości. Zdefiniowanie wielkości piłeczki jako stałej pomaga zrozumieć kod, w którym ta wartość musi być użyta. Kolejną zaletą takiego podejścia jest to, że jeśli zmienisz zdanie i zdecydujesz, że piłeczka powinna być większa lub mniejsza, odpowiednią zmianę będziesz musiał wprowadzić tylko w jednym miejscu kodu: w deklaracji stałej `BALL_SIZE`.

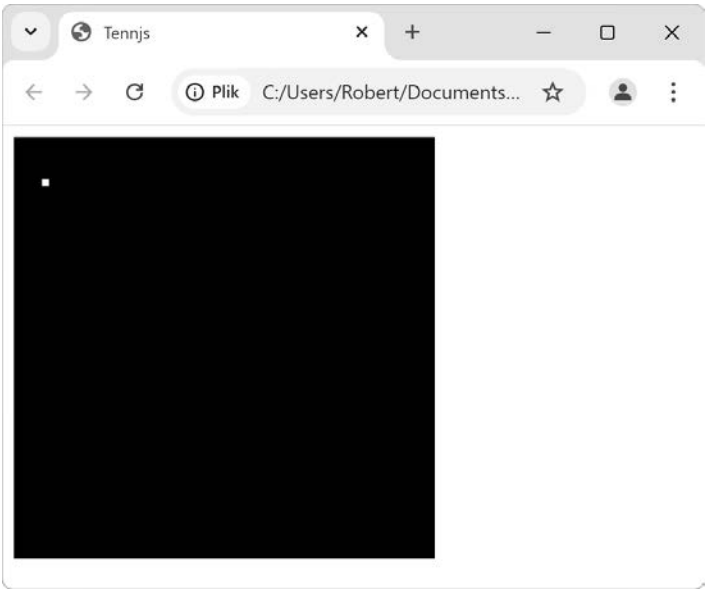
Położenie piłeczki jest śledzone za pomocą obiektu zawierającego jej współrzędne X i Y, utworzonego z użyciem składni literału obiektu ❶. W poprzednim rozdziale użyliśmy oddzielnych zmiennych dla współrzędnych X i Y rysowanego koła. Jednak znacznie bardziej przejrzyste będzie przechowywanie dwóch powiązanych ze sobą zmiennych jako pojedynczego obiektu, zwłaszcza że program będzie nieco dłuższy i zdecydowanie znacznie bardziej skomplikowany.

Odśwież stronę *index.html*, a zobaczysz białą piłeczkę znajdującą się w pobliżu lewego górnego rogu płótna, jak pokazałem na rysunku 10.2.

Obecnie piłeczka jest nieruchoma, ale wkrótce utworzymy kod, który pozwoli wprawić ją w ruch.

Refaktoryzacja

Przystępujemy do prostej **refaktoryzacji**, która w kontekście tworzenia oprogramowania oznacza modyfikację pewnego kodu bez zmiany sposobu jego działania. Celem refaktoryzacji jest zwykle ułatwienie zrozumienia kodu bądź jego uaktualnienie. Gdy kod projektu stanie się bardziej skomplikowany, refaktoryzacja pomoże w zachowaniu właściwej jego organizacji.



Rysunek 10.2. Piłeczka wyświetlona na płótnie

W naszej grze rysowanie na płótnie będzie odbywało się wielokrotnie, a nie tylko raz. W rzeczywistości ostatecznym celem jest ponowne rysowanie na płótnie co 30 ms, aby wywołać wrażenie ruchu w grze. To zadanie można sobie ułatwić poprzez refaktoryzację kodu i umieszczenie wszystkich poleceń odpowiedzialnych za rysowanie w nowej funkcji o nazwie `draw`. Dzięki temu będzie można po prostu wywołać tę funkcję za każdym razem, gdy pojawi się konieczność rysowania na płótnie.

W pliku *script.js* wprowadź zmiany przedstawione na listingu 10.4.

Listing 10.4. Refaktoryzacja kodu przeznaczonego do rysowania na płótnie

```
let canvas = document.querySelector("#canvas");
let ctx = canvas.getContext("2d");
let width = canvas.width;
let height = canvas.height;

const BALL_SIZE = 5;
let ballPosition = { x: 20, y: 30 };

function draw() {    ❶
  ctx.fillStyle = "black";
  ctx.fillRect(0, 0, width, height);

  ctx.fillStyle = "white";
  ctx.fillRect(ballPosition.x, ballPosition.y, BALL_SIZE, BALL_SIZE);
}

draw();    ❷
```

Jedyna wprowadzona tutaj zmiana polega na grupowaniu całego kodu obsługującego rysowanie na płótnie w pojedynczej funkcji o nazwie `draw` ❶, która zostaje natychmiast wywołana ❷. Ponieważ to jest tylko refaktoryzacja, rzeczywisty sposób działania kodu nie został zmieniony. Po odświeżeniu strony *index.html* w przeglądarce internetowej przekonasz się, że wszystko wygląda i działa jak wcześniej.

Pętla gry

Praktycznie każda gra zwiera tzw. **pętlę gry**, która koordynuje wszystko, co zachodzi w trakcie poszczególnych klatek gry. Taka pętla jest podobna do pętli animacji, np. przedstawionej w poprzednim rozdziale, choć ma dodatkową logikę. Spójrz teraz na ogólną postać pętli gry, z którą można się spotkać w większości gier.

1. Usunięcie zawartości płótna.
2. Narysowanie obrazu.
3. Pobranie danych wejściowych od gracza.
4. Uaktualnienie stanu.
5. Sprawdzenie pod kątem kolizji.
6. Odczekanie krótkiego czasu.
7. Powtórzenie.

Pobieranie danych wejściowych od gracza (lub graczy) i reagowanie na nie to najważniejsza cecha odróżniająca grę od animacji. **Wykrywanie kolizji** to kolejny ważny aspekt w większości gier: sprawdzenie, czy dwa obiekty gry zetknęły się ze sobą, i odpowiednia reakcja na to. Wykrywanie kolizji to mechanizm, dzięki któremu bohater gry nie może przechodzić przez ściany bądź przejeżdżać pojazdem przez inny pojazd. W omawianym przykładzie ten mechanizm spowoduje, że piłeczka będzie odbijała się od ścian i paletek. Poza danymi wejściowymi pochodzącymi od gracza i możliwością wykrywania kolizji kroki wymienione w pętli gry są mniej więcej takie same jak w pętli animacji: usunięcie zawartości płótna, narysowanie obrazu, uaktualnienie stanu gry w celu przesunięcia obiektów do ich nowego położenia, krótka przerwa i powtórzenie procedury.

Zamiast podejmować próbę utworzenia od razu całej pętli gry, opracujemy ją stopniowo. Uaktualnij skrypt *script.js* do postaci pokazanej na listingu 10.5. Mamy tutaj początek pętli gry w naszej grze. Ten kod powoduje przeniesienie piłeczki (tzn. uaktualnienie jej stanu), ponowne narysowanie zawartości płótna, krótką przerwę i powtórzenie procedury od początku.

Listing 10.5. Pętla gry

```
  --cięcie--
const BALL_SIZE = 5;
let ballPosition = { x: 20, y: 30 };

let xSpeed = 4;    ❶
```

```
let ySpeed = 2;

function draw() {
  ctx.fillStyle = "black";
  ctx.fillRect(0, 0, width, height);

  ctx.fillStyle = "white";
  ctx.fillRect(ballPosition.x, ballPosition.y, BALL_SIZE, BALL_SIZE);
   }

function update() {    ❷
  ballPosition.x += xSpeed;
  ballPosition.y += ySpeed;
   }

function gameLoop() {    ❸
  draw();
  update();

  // Ponowne wywołanie tej funkcji po upływie określonego czasu
  setTimeout(gameLoop, 30);
}

gameLoop();  ❹
```

Pierwsza wprowadzona zmiana polega na inicjalizacji dwóch nowych zmiennych ❶, `xSpeed` i `ySpeed`, które będą kontrolowały szybkość poruszania się piłeczki w poziomie oraz w pionie. Nowa funkcja, `update` ❷, używa tych zmiennych do uaktualnienia położenia piłeczki. W trakcie każdej klatki piłeczka przesunie się o `xSpeed` pikseli wzdłuż osi X oraz o `ySpeed` pikseli wzdłuż osi Y. Na początku te dwie zmienne mają wartość, odpowiednio, `4` i `2`, więc w każdej klatce piłeczka przesunie się o cztery piksele w prawo i dwa piksele w dół.

Funkcja `gameLoop` ❸ najpierw wywołuje funkcje `draw` i `update`. Następnie wywołuje polecenie `setTimeout(gameLoop, 30)`, które nakazuje ponowne wywołanie funkcji `gameLoop` po upływie 30 ms. To praktycznie identyczna technika jak użyte w poprzednim rozdziale wywołanie `setInterval`. Przypomnij sobie, że funkcja `setTimeout` będzie wywoływała wskazaną funkcję po upływie podanego czasu, podczas gdy `setInterval` nieustannie wywołuje wskazaną w niej funkcję. W grze użyliśmy `setTimeout`, aby zachować większą kontrolę nad przeprowadzaniem pętli. W dalszej części rozdziału do kodu dodamy pewną logikę warunkową, która będzie wywoływała `setTimeout` bądź kończyła grę.

Zwróć uwagę na rozpoczynający się od dwóch ukośników (//) wiersz, który znajduje się przed wywołaniem `setTimeout`. To przykład *komentarza*, czyli osadzonych w pliku programu informacji dla Ciebie (bądź innych osób czytających ten kod źródłowy). W trakcie wykonywania programu JavaScript ignoruje wszystkie wiersze rozpoczynające się od znaków // (to, co znajduje się przez znakami //, wciąż będzie przetwarzane jako kod JavaScript). Dlatego też tego rodzaju komentarzy można używać do wyjaśnienia sposobu działania kodu, podkreślenia ważnych funkcji, wskazania na konieczność dopracowania pewnego fragmentu itd., bez wpływania na funkcjonalność programu.

Na końcu skryptu znajduje się wywołanie funkcji `gameLoop` ❹, które wprawia grę w ruch. Skoro funkcja `gameLoop` kończy się wywołaniem `setTimeout`, będzie nieustannie wywoływana co 30 ms. Odśwież stronę w przeglądarce internetowej, a zobaczysz, że piłeczka porusza się na ukos, do dołu oraz w prawą stronę, podobnie jak animacja utworzona w poprzednim rozdziale.

Odbijanie się piłeczki

W poprzednim podrozdziale zajęliśmy się wprawieniem piłeczki w ruch, przy czym poruszała się ona jedynie od lewej krawędzi płótna. Teraz dowiesz się, jak odbijać piłeczkę od krawędzi płótna, pod odpowiednim kątem — to będzie nasz pierwszy kod w zakresie wykrywania kolizji. Plik *script.js* uaktualnij do postaci przedstawionej na listingu 10.6. Nowy kod to definicja funkcji `checkCollision`.

Listing 10.6. Wykrywanie kolizji piłeczki ze ścianami

```
--cięcie--
function update() {
  ballPosition.x += xSpeed;
  ballPosition.y += ySpeed;
}

function checkCollision() {
  let ball = {    ❶
    left: ballPosition.x,
    right: ballPosition.x + BALL_SIZE,
    top: ballPosition.y,
    bottom: ballPosition.y + BALL_SIZE
  }

  if (ball.left < 0 || ball.right > width) {    ❷
    xSpeed = -xSpeed;
  }
  if (ball.top < 0 || ball.bottom > height) {    ❸
    ySpeed = -ySpeed;
  }
}

function gameLoop() {
  draw();
  update();
  checkCollision();  ❹

// Ponowne wywołanie tej funkcji po upływie określonego czasu
  setTimeout(gameLoop, 30);
}

gameLoop();
```

Zadaniem nowej funkcji, `checkCollision`, jest sprawdzenie, czy piłeczka zderzyła się z jedną z czterech ścian tworzonych przez krawędzie płótna. Jeżeli tak, nastąpi odpowiednie uaktualnienie wartości zmiennych `xSpeed` i `ySpeed`, aby piłeczka odbiła się od ściany. Trzeba zacząć od obliczenia wartości dla krawędzi piłeczki. Musimy wiedzieć, gdzie znajdują się krawędzie: lewa, prawa, górna i dolna piłeczki, i dopiero wtedy możemy ustalić, czy przekroczyły one granice obszaru rozgrywki. Te wartości grupujemy w obiekcie `ball` ❶, który ma właściwości `left`, `right`, `top` i `bottom`. Określenie lewej i górnej krawędzi piłeczki jest łatwe: odpowiadają one właściwościom, odpowiednio, `ballPosition.x` i `ballPosition.y`. Aby otrzymać krawędź prawą i dolną, trzeba dodać wartość `BALL_SIZE` do `ballPosition.x` i `ballPosition.y`. To jeden ze wspomnianych wcześniej przypadków, w którym użyteczna jest możliwość uzyskania dostępu do stałej określającej wielkość piłeczki.

W kolejnym kroku odbywa się rzeczywiste wykrywanie kolizji. Jeżeli lewa krawędź piłeczki jest w położeniu mniejszym niż `0` lub prawa krawędź jest w położeniu większym niż szerokość płótna ❷, wówczas wiemy, że piłeczka uderzyła w, odpowiednio, lewą lub prawą ścianę. W obu przypadkach działanie matematyczne, które trzeba wykonać, jest dokładnie takie samo: nowa wartość `xSpeed` musi mieć przeciwny znak do obecnej (to znaczy wartość jest *negowana*). Na przykład gdy piłeczka po raz pierwszy zderzy się z prawą krawędzią, wartość `xSpeed` zostanie zmieniona z `4` na `-4`. W tym czasie wartość `ySpeed` pozostaje niezmieniona. W efekcie piłeczka nadal porusza się w dół ekranu, z tą samą szybkością, ale teraz ruch odbywa się w lewą stronę zamiast w prawą.

Ten sam rodzaj operacji sprawdzenia odbywa się dla górnej krawędzi piłeczki zderzającej się z górną ścianą i dolnej krawędzi piłeczki zderzającej się z dolną ścianą ❸. W obu tych przypadkach negowana jest wartość `ySpeed`: po uderzeniu w górną ścianę wartość zmienia się z `2` na `-2`, natomiast po uderzeniu w dolną ścianę następuje zmiana z wartości `-2` na `2`.

Zmianą w kodzie jest dodanie wywołania metody `checkCollision` do listy metod w funkcji `gameLoop` ❹. Teraz po odświeżeniu strony *index.html* w przeglądarce internetowej zobaczysz, że piłeczka nieustannie odbija się od ścian obszaru rozgrywki.

Jeżeli zwracasz dokładną uwagę na szczegóły, to zauważysz, że piłeczka nie powinna odbijać się od lewej i prawej ściany. Po dodaniu paletek zmodyfikujemy ten kod wykrywania kolizji, aby piłeczka odbijała się jedynie od górnej i dolnej ściany, zaś kolizja ze ścianą boczną oznaczała zdobycie punktu przez jednego z graczy.

Paletki

Następne zadanie polega na narysowaniu dwóch paletek. To wymaga wprowadzenia pewnych nowych stałych określających wymiary paletek oraz ich położenie względem ścian płótna, a także zmiennych definiujących pionowe położenie paletek. (Paletki mogą poruszać się jedynie w górę i w dół, a nie na boki, więc tylko ich położenie pionowe musi być zapisywane w zmiennych). Plik *script.js* uaktualnij do postaci pokazanej na listingu 10.7.

Listing 10.7. Zdefiniowanie paletek w grze

```
--cięcie--
let xSpeed = 4;
let ySpeed = 2;

const PADDLE_WIDTH = 5;
const PADDLE_HEIGHT = 20;
const PADDLE_OFFSET = 10;

let leftPaddleTop = 10;
let rightPaddleTop = 30;

function draw() {
--cięcie--
```

Przede wszystkim dodaliśmy stałe definiujące paletki. `PADDLE_WIDTH` i `PADDLE_HEIGHT` definiują wymiary paletki na 5 pikseli szerokości i 20 pikseli wysokości. Z kolei `PADDLE_OFFSET` to odległość paletki od lewej bądź prawej krawędzi obszaru rozgrywki.

Zmienne `leftPaddleTop` i `rightPaddleTop` określają aktualne pionowe położenie górnej krawędzi paletki. Ostatecznie wartość `leftPaddleTop` będzie kontrolowana przez komputer za pomocą utworzonej później funkcji, aby paletka komputera podążała za piłeczką. Z kolei wartość `rightPaddleTop` będzie uaktualniana, gdy gracz porusza myszą. Na początek tym zmiennym zostały przypisane wartości, odpowiednio, `10` i `30`.

Trzeba uaktualnić kod funkcji `draw`, aby wyświetlała paletki na podstawie zdefiniowanych przed chwilą informacji. Do kodu dodałem także komentarze wyjaśniające sposób działania funkcji `draw`. Zmodyfikuj kod pliku *script.js* do postaci pokazanej na listingu 10.8.

Listing 10.8. Rysowanie paletek na płótnie

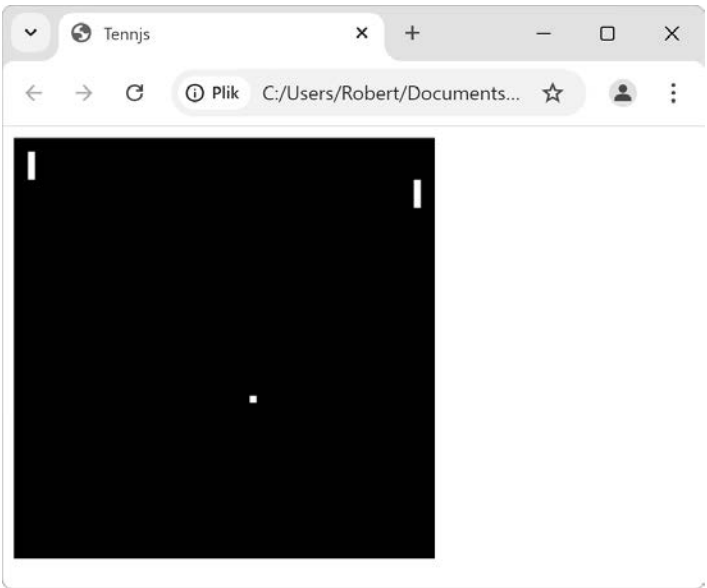
```
--cięcie--
function draw() {
  // Wypełnienie płótna kolorem czarnym
  ctx.fillStyle = "black";
  ctx.fillRect(0, 0, width, height);

  // Wszystko pozostałe będzie w kolorze białym
  ctx.fillStyle = "white";

  // Narysowanie piłeczki
  ctx.fillRect(ballPosition.x, ballPosition.y, BALL_SIZE, BALL_SIZE);

  // Narysowanie paletek
  ctx.fillRect(    ❶
    PADDLE_OFFSET,
    leftPaddleTop,
    PADDLE_WIDTH,
    PADDLE_HEIGHT
  );
  ctx.fillRect(    ❷
    width - PADDLE_WIDTH - PADDLE_OFFSET,
    rightPaddleTop,
```

```
    PADDLE_WIDTH,
    PADDLE_HEIGHT
  );
}

function update() {
--cięcie--
```

Pomijając dodatkowe komentarze pomagające udokumentować tworzony program, w nowym kodzie znajdują się dwa wywołania funkcji `fillRect`. Pierwsze odpowiada za narysowanie lewej paletki ❶, natomiast drugie za narysowanie prawej ❷. Argumenty tej funkcji podzieliłem na wiele wierszy, ponieważ ich identyfikatory są długie. Pamiętaj, że parametrami funkcji `fillRect` są `x`, `y`, `width` i `height`, gdzie `x` oraz `y` to współrzędne lewego górnego rogu prostokąta. Współrzędna X lewej paletki to `PADDLE_OFFSET`, ponieważ ta wartość jest używana do określenia odległości paletki od lewej krawędzi płótna. Z kolei współrzędna Y lewej paletki to po prostu wartość `leftPaddleTop`. Argumenty `width` i `height` mają przypisane wartości stałych, odpowiednio, `PADDLE_WIDTH` i `PADDLE_HEIGHT`.

Narysowanie prawej paletki okazuje się nieco bardziej skomplikowane. Aby pobrać współrzędną X lewego górnego rogu paletki, trzeba wziąć szerokość płótna oraz odjąć od niej szerokość paletki i wartość `PADDLE_OFFSET`. Skoro szerokość płótna wynosi 500, a szerokość paletki 10, podobnie jak wartość `PADDLE_OFFSET`, to współrzędna X prawej paletki ma wartość 480.

Po odświeżeniu strony *index.html* zobaczysz nie tylko odbijającą się piłeczkę, ale również dwie paletki, jak pokazałem na rysunku 10.3.



Rysunek 10.3. Paletki i piłeczka

Zauważ, że teraz piłeczka przechodzi przez paletki, ponieważ jeszcze nie zdefiniowaliśmy wykrywania kolizji dla paletek. Tym zajmiemy się nieco później w rozdziale.

Poruszanie paletkami pod wpływem działań użytkownika

Paletki są rysowane w położeniu pionowym wskazywanym przez zmienne `leftPaddleTop` i `rightPaddleTop`. Dlatego też, aby paletkami poruszać w górę oraz w dół, trzeba po prostu uaktualnić wartości tych zmiennych. Obecnie są połączone jedynie z prawą paletką, która jest kontrolowana przez człowieka.

Aby umożliwić graczowi kontrolowanie prawej paletki, do pliku *script.js* dodamy procedurę obsługi zdarzeń, która będzie nasłuchiwała zdarzeń `mousemove`. Kod tej procedury zamieściłem na listingu 10.9.

Listing 10.9. Dodawanie procedury obsługi zdarzeń przeznaczonej do obsługi prawej paletki

```
--cięcie--
let leftPaddleTop = 10;
let rightPaddleTop = 30;

document.addEventListener("mousemove", e => {
  rightPaddleTop = e.y - canvas.offsetTop;
});

function draw() {
--cięcie--
```

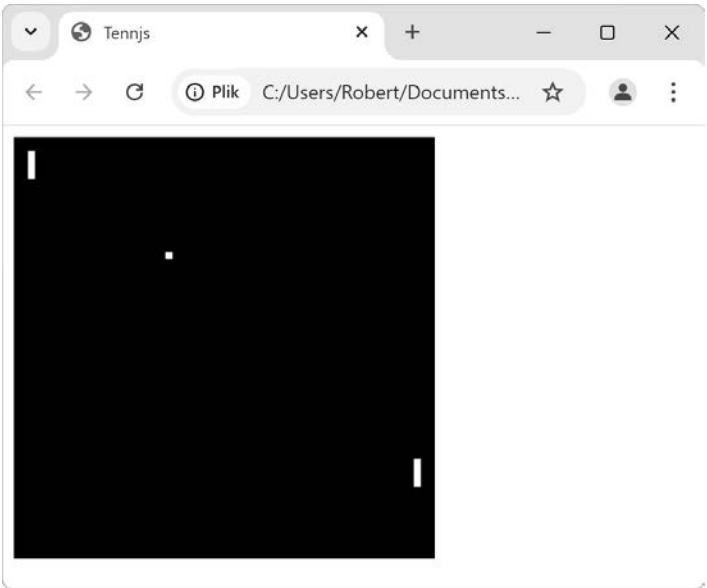
W tym kodzie został zastosowany ten sam wzorzec procedury obsługi zdarzeń, który znasz z rozdziału 8. Metody `document.addEventListener` użyliśmy w celu dodania procedury obsługi zdarzeń `mousemove`. Po wykryciu takiego zdarzenia funkcja procedury obsługi uaktualnia wartość `rightPaddleTop` na podstawie współrzędnej Y pochodzącej ze zdarzenia `mousemove` (`e.y`). Wartość tej współrzędnej jest podawana względem górnej krawędzi strony, a nie płótna, więc od otrzymanej współrzędnej Y trzeba odjąć wartość `canvas.offsetTop` (odległość od górnej krawędzi płótna do górnej krawędzi strony). Tak przypisana wartość `rightPaddleTop` zostanie obliczona na podstawie odległości kursora myszy od górnej krawędzi płótna, zaś paletka będzie poprawnie podążała za kursorem myszy.

Odśwież stronę *index.html*, a przekonasz się, że teraz prawa paletka porusza się pionowo zgodnie z ruchami kursora myszy. Na rysunku 10.4 pokazałem, jak powinien wyglądać bieżący obszar rozgrywki.

Budowana przez nas gra wreszcie zaczęła być interaktywna, a gracz ma pełną kontrolę nad położeniem prawej paletki.

Wykrywanie kolizji paletki

Następnym krokiem jest dodanie mechanizmu wykrywania kolizji paletek. W grze musimy wiedzieć, czy piłeczka uderzyła paletkę, i jeśli tak, w odpowiedni sposób ją odbić. To będzie wymagało użycia większej ilości kodu, więc zostanie on podzielony na kilka fragmentów.



Rysunek 10.4. Prawa paletka podąża za kursorem myszy

Pierwszym zadaniem jest utworzenie obiektów definiujących cztery krawędzie dwóch paletek, podobnie jak to miało miejsce w przypadku obiektu dla piłeczki utworzonego w kodzie na listingu 10.6. Zmiany niezbędne do wprowadzenia w pliku *script.js* przedstawiłem na listingu 10.10.

Listing 10.10. Zdefiniowanie krawędzi paletek

```
--cięcie--
function checkCollision() {
  let ball = {
    left: ballPosition.x,
    right: ballPosition.x + BALL_SIZE,
    top: ballPosition.y,
    bottom: ballPosition.y + BALL_SIZE
  }

  let leftPaddle = {
    left: PADDLE_OFFSET,
    right: PADDLE_OFFSET + PADDLE_WIDTH,
    top: leftPaddleTop,
    bottom: leftPaddleTop + PADDLE_HEIGHT
  };

  let rightPaddle = {
    left: width - PADDLE_WIDTH - PADDLE_OFFSET,
    right: width - PADDLE_OFFSET,
    top: rightPaddleTop,
```

```
    bottom: rightPaddleTop + PADDLE_HEIGHT
  };

  if (ball.left < 0 || ball.right > width) {
--cięcie--
```

Obiekty `leftPaddle` i `rightPaddle` zawierają informacje o krawędziach odpowiednich paletek, umieszczone w czterech właściwościach: `left`, `right`, `top` i `bottom`. Podobnie jak miało to miejsce w kodzie przedstawionym na listingu 10.8, określenie krawędzi prawej paletki wymaga nieco więcej obliczeń, ponieważ pod uwagę trzeba wziąć szerokość płótna, przesunięcie paletki i jej szerokość.

Potrzebna jest również funkcja, nazwaliśmy ją `checkPaddleCollision`, która będzie pobierać obiekt piłeczki oraz jednej z paletek, a następnie zwróci `true`, jeśli piłeczka wejdzie w kolizję z daną paletką. Definicja tej funkcji znajduje się na listingu 10.11.

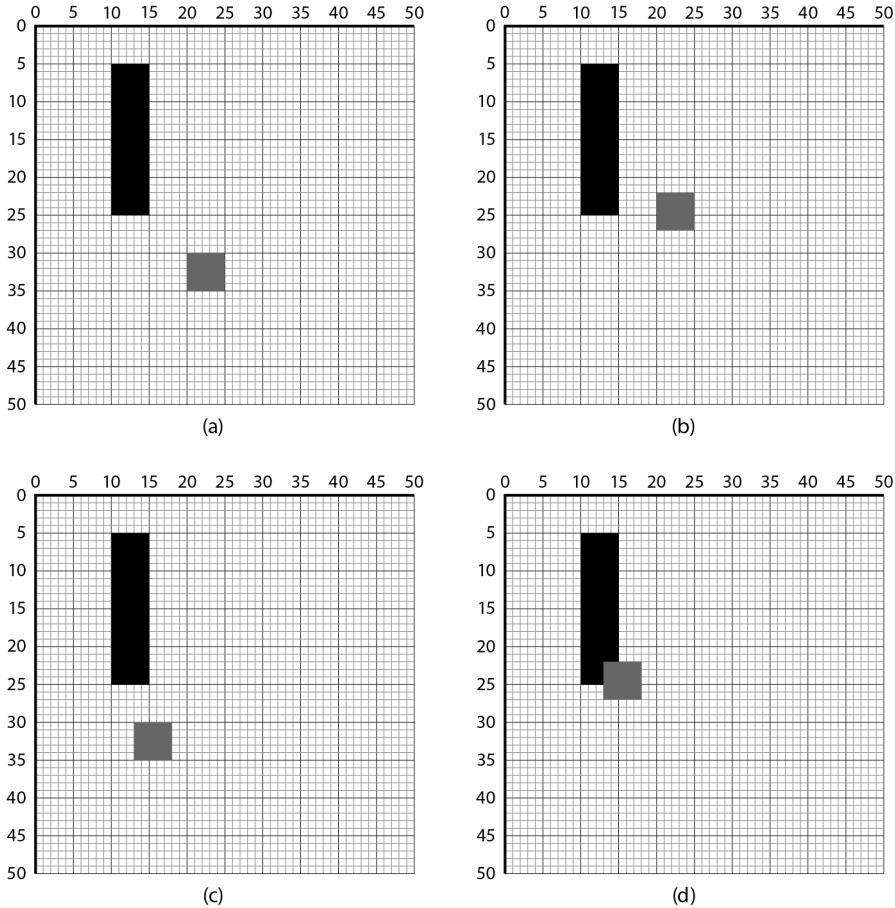
Listing 10.11. Funkcja checkPaddleCollision

```
--cięcie--
function update() {
  ballPosition.x += xSpeed;
  ballPosition.y += ySpeed;
}

function checkPaddleCollision(ball, paddle) {
  // Sprawdzenie, czy paletka i piłeczka nachodzą na siebie w pionie oraz w poziomie
  return (
    ball.left   < paddle.right &&
    ball.right  > paddle.left &&
    ball.top    < paddle.bottom &&
    ball.bottom > paddle.top
  );
}

function checkCollision() {
--cięcie--
```

Ta funkcja będzie wywołana razem ze zdefiniowanymi wcześniej obiektami piłeczki i poszczególnych paletek. Używa długiego wyrażenia boolowskiego, składającego się z czterech podwyrażeń połączonych operatorami logicznymi `&&`. Wartością zwrotną tej funkcji jest `true` tylko wtedy, gdy wszystkie cztery podwyrażenia również są spełnione. (Dodałem nieco wolnego miejsca w tych podwyrażeniach, aby wyrównać pionowo operandy, ponieważ wówczas kod jest łatwiejszy w odczycie). Ujmując rzecz najprościej — to wyrażenie działa tak:

1. Lewa krawędź piłeczki musi być po lewej stronie prawej krawędzi paletki.
2. Prawa krawędź piłeczki musi być po prawej stronie lewej krawędzi paletki.
3. Górna krawędź piłeczki musi być powyżej dolnej krawędzi paletki.
4. Dolna krawędź piłeczki musi być poniżej górnej krawędzi paletki.

Jeżeli dwa pierwsze warunki są spełnione, wówczas piłeczka nachodzi na paletkę poziomo, natomiast spełnienie dwóch ostatnich warunków oznacza, że piłeczka nachodzi na paletkę pionowo. Piłeczka wchodzi w kolizję z paletką tylko w przypadku spełnienia wszystkich czterech warunków. W sposób graficzny zilustrowałem to na rysunku 10.5.



Rysunek 10.5. Warunki wykrywania kolizji w naszej grze

Na rysunku 10.5 pokazałem cztery możliwe scenariusze, pod kątem których można przeprowadzić sprawdzenie. W tych wszystkich scenariuszach paletka ma krawędzie w następujących punktach: `{ left: 10, right: 15, top: 5, bottom: 25 }`.

Na rysunku 10.5(a) piłeczka ma krawędzie w następujących punktach: `{ left: 20, right: 25, top: 30, bottom: 35 }`. W tym przypadku warunek `ball.left < paddle.right` jest niespełniony (lewa krawędź piłeczki nie jest po lewej stronie prawej krawędzi paletki), ale spełniony jest warunek `ball.right > paddle.left`. Podobnie warunek `ball.top < paddle.bottom` jest niespełniony, a spełniony jest `ball.bottom > paddle.top`. Zatem piłeczka ani poziomo, ani pionowo nie przecina się z paletką.

Na rysunku 10.5(b) piłeczka ma krawędzie w następujących punktach: `{ left: 20, right: 25, top: 22, bottom: 27 }`. Tym razem warunki `ball.top < paddle.bottom` i `ball.bottom > paddle.top` są spełnione, co oznacza, że piłeczka pionowo przecina się z paletką, ale nie poziomo.

Na rysunku 10.5(c) piłeczka ma krawędzie w następujących punktach: `{ left: 13, right: 18, top: 30, bottom: 35 }`. W tym przypadku piłeczka poziomo przecina się z paletką, ale nie pionowo.

Wreszcie na rysunku 10.5(d) piłeczka ma krawędzie w następujących punktach: `{ left: 13, right: 18, top: 22, bottom: 27 }`. Teraz piłeczka poziomo i pionowo przecina się z paletką. Skoro wszystkie cztery podwyrażenia są prawdziwe, funkcja `checkPaddleCollision` zwraca `true`.

Nadeszła pora na rzeczywiste wywołanie funkcji `checkPaddleCollision` z poziomu funkcji `checkCollision`, jednokrotnie dla każdej paletki, oraz obsłużenie sytuacji, w której wartością zwrotną funkcji jest `true`. Odpowiedni kod zamieściłem na listingu 10.12.

Listing 10.12. Sprawdzenie pod kątem kolizji piłeczki z paletkami

```
--cięcie--
  let rightPaddle = {
    left: width - PADDLE_WIDTH - PADDLE_OFFSET,
    right: width - PADDLE_OFFSET,
    top: rightPaddleTop,
    bottom: rightPaddleTop + PADDLE_HEIGHT
  };

  if (checkPaddleCollision(ball, leftPaddle)) {
    // Doszło do kolizji z lewą paletką
    xSpeed = Math.abs(xSpeed);  ❶
  }

  if (checkPaddleCollision(ball, rightPaddle)) {
    // Doszło do kolizji z prawą paletką
    xSpeed = -Math.abs(xSpeed);  ❷
  }

  if (ball.left < 0 || ball.right > width) {
    xSpeed = -xSpeed;
  }
  if (ball.top < 0 || ball.bottom > height) {
    ySpeed = -ySpeed;
  }
}
--cięcie--
```

Pamiętaj, że funkcja `checkPaddleCollision` pobiera obiekty przedstawiające piłeczkę i paletkę oraz zwraca wartość `true`, jeśli one na siebie nachodzą. Jeżeli wywołanie `check↪PaddleCollision(ball, leftPaddle)` zwraca wartość `true`, wówczas `xSpeed` przypisujemy wartość `Math.abs(xSpeed)` ❶, co odpowiada wartości `4`, ponieważ w naszej grze `xSpeed` może przyjmować jedynie wartości `4` (podczas ruchu w prawą stronę) lub `-4` (podczas ruchu w lewą stronę).

Być może zastanawiasz się nad tym, dlaczego po prostu nie negujemy wartości `xSpeed`, podobnie jak to miało miejsce wcześniej w kodzie wykrywającym pionową kolizję ze ścianą. Użycie tutaj wartości bezwzględnej jest pewnego rodzaju sztuczką, aby uniknąć wielu kolizji, które mogłyby doprowadzić do odbijania się piłeczki „wewnątrz" paletki. Istnieje prawdopodobieństwo, że jeśli piłeczka uderzy w prawy punkt znajdujący się na końcu paletki, to zostanie odbita, ale w trakcie następnej klatki znów zostanie wykryta kolizja z tą samą paletką. W przypadku negowania wartości `xSpeed` dojdzie do odbijania piłeczki w paletce. Wymuszenie, aby uaktualniona wartość `xSpeed` była dodatnia, gwarantuje, że w wyniku kolizji z lewą paletką piłeczka zawsze będzie poruszała się w prawą stronę.

Podobne rozwiązanie zostało zastosowane podczas obsługi kolizji z prawą paletką. W tym przypadku kolizja powoduje przypisanie `xSpeed` wartości `-Math.abs(xSpeed)` ❷, czyli w tym przypadku `-4`, co oznacza, że piłeczka zostanie odbita w lewą stronę.

Odśwież stronę *index.html* i spróbuj poruszyć myszą prawą paletką w taki sposób, aby odbić piłeczkę. W tym momencie mamy zaimplementowaną obsługę odbijania piłeczki paletkami. Wprawdzie piłeczka jeszcze odbija się od ścian bocznych, ale wkrótce się tym zajmiemy.

Odbicie piłeczki w pobliżu końca paletki

Na początku rozdziału wspomniałem, że w grze Pong można zmienić kąt odbicia piłeczki w przypadku jej uderzenia w pobliżu górnej lub dolnej krawędzi paletki. Teraz zajmiemy się implementacją tej funkcjonalności. Trzeba zacząć od dodania nowej funkcji o nazwie `adjustAngle` tuż przed `checkCollision`. Będzie ona sprawdzała, czy piłeczka znajduje się w pobliżu górnej lub dolnej krawędzi paletki, i jeśli tak, zmodyfikuje wartość `ySpeed`. Kod nowej funkcji zamieściłem na listingu 10.13.

Listing 10.13. Dostosowanie kąta odbicia piłeczki

```
--cięcie--
function adjustAngle(distanceFromTop, distanceFromBottom) {
  if (distanceFromTop < 0) {  ❶
    // Jeżeli piłeczka uderzy w pobliżu górnej krawędzi paletki, należy zmniejszyć wartość ySpeed
    ySpeed -= 0.5;
  } else if (distanceFromBottom < 0) {  ❷
    // Jeżeli piłeczka uderzy w pobliżu dolnej krawędzi paletki, należy zwiększyć wartość ySpeed
    ySpeed += 0.5;
  }
}

function checkCollision() {
--cięcie--
```

Funkcja `adjustAngle` ma dwa parametry, `distanceFromTop` i `distanceFromBottom`. Przedstawiają one odległość, odpowiednio, od górnej krawędzi piłeczki do górnej krawędzi paletki oraz od dolnej krawędzi paletki do dolnej krawędzi piłeczki. Ta funkcja najpierw sprawdza, czy wartość `distanceFromTop` jest mniejsza niż `0` ❶. Jeżeli tak, w trakcie kolizji górna krawędź piłeczki znajduje się ponad górną krawędzią paletki i tak właśnie definiujemy

znalezienie się piłeczki w pobliżu górnej krawędzi paletki. W takim przypadku wartość `ySpeed` zostaje zmniejszona o `0.5`. Jeżeli piłeczka porusza się w dół ekranu, gdy znajdzie się w pobliżu górnej krawędzi paletki, wartość `ySpeed` jest dodatnia, więc odjęcie od niej `0.5` zmniejsza szybkość w pionie. Na przykład na początku gry wartość `ySpeed` wynosi `2`. Jeżeli umieścisz paletkę w taki sposób, aby piłeczka uderzyła w pobliżu jej górnej krawędzi, po odbiciu piłeczki wartość `ySpeed` zmniejszy się do `1.5`, zmniejszając tym samym kąt odbicia. Natomiast jeśli piłeczka porusza się w górę ekranu, wówczas wartość `ySpeed` jest ujemna. W takim przypadku odjęcie `0.5` po uderzeniu piłeczki w pobliżu górnej krawędzi paletki spowoduje zwiększenie szybkości piłeczki w pionie. Na przykład wartość `ySpeed` zmieni się z `-2` na `-2.5`.

Jeżeli piłeczka uderzy paletkę w pobliżu jej dolnej krawędzi ❷, będziemy mieli do czynienia z sytuacją przeciwną. Wówczas dodanie `0.5` do wartości `ySpeed` spowoduje zwiększenie (w przypadku ruchu w dół ekranu) lub zmniejszenie (w przypadku ruchu w górę ekranu) szybkości piłeczki w pionie.

Konieczne jest uaktualnienie funkcji `checkCollision`, aby wywoływała nową funkcję `adjustAngle` w trakcie wykonywania logiki wykrywania kolizji piłeczki z paletkami. Zmiany niezbędne do wprowadzenia przedstawiłem na listingu 10.14.

Listing 10.14. Wywołanie funkcji adjustAngle

```
--cięcie--
  let rightPaddle = {
    left: width - PADDLE_WIDTH - PADDLE_OFFSET,
    right: width - PADDLE_OFFSET,
    top: rightPaddleTop,
    bottom: rightPaddleTop + PADDLE_HEIGHT
  };

  if (checkPaddleCollision(ball, leftPaddle)) {
    // Doszło do kolizji z lewą paletką
    let distanceFromTop = ball.top - leftPaddle.top;
    let distanceFromBottom = leftPaddle.bottom - ball.bottom;
    adjustAngle(distanceFromTop, distanceFromBottom);
    xSpeed = Math.abs(xSpeed);
  }

  if (checkPaddleCollision(ball, rightPaddle)) {
    // Doszło do kolizji z prawą paletką
    let distanceFromTop = ball.top - rightPaddle.top;
    let distanceFromBottom = rightPaddle.bottom - ball.bottom;
    adjustAngle(distanceFromTop, distanceFromBottom);
    xSpeed = -Math.abs(xSpeed);
  }
--cięcie--
```

W poleceniu `if` dla każdej paletki deklarujemy wartości `distanceFromTop` i `distanceFrom`
`↪Bottom`, czyli argumenty niezbędne podczas wywoływania funkcji `adjustAngle`. Następnie wywołujemy funkcję `adjustAngle` tuż przed uaktualnieniem wartości `xSpeed` jak wcześniej.

Wypróbuj grę i sprawdź, czy potrafisz uderzyć piłeczkę niemalże przy krawędzi paletki.

WYPRÓBUJ SAMODZIELNIE

10.1. Uderzenie krawędzią paletki może być trudne. Aby to sobie ułatwić, spróbuj zmniejszyć szybkość gry poprzez zwiększenie czasu w `setTimeout` np. z 30 ms do 60 ms. Inną możliwością jest zmiana definicji tego, co uznajemy za „w pobliżu górnej krawędzi" lub „w pobliżu dolnej krawędzi" paletki. Zamiast wyrażenia `distanceFromTop < 0` można użyć np. `distanceFromTop < 5` i wówczas sprawdzamy, czy piłeczka znajduje się w odległości mniejszej niż 5 pikseli od górnej krawędzi paletki.

10.2. Nie zawsze będzie oczywiste, że doszło do uderzenia w pobliżu górnej bądź dolnej krawędzi paletki, ponieważ zmiana wartości `ySpeed` jest bardzo mała. Aby dostarczyć informacji zwrotnych o tym, co się faktycznie dzieje, gdy piłeczka uderza w paletkę, w funkcji `adjustAngle` można umieścić polecenie, które będzie wyświetlało w konsoli odpowiednie informacje. Na przykład na początku funkcji może znaleźć się następujące polecenie:

```
console.log(`top: ${distanceFromTop}, bottom: ${distanceFromBottom}`);
```

W ten sposób konsola będzie wyświetlała informacje na temat odległości piłeczki od górnej i dolnej krawędzi paletki za każdym razem, gdy piłeczka uderzy w paletkę. Innym rozwiązaniem, które może pomóc, jest dodanie poleceń wyświetlających komunikaty do dwóch konstrukcji warunkowych w funkcji `adjustAngle`, jak pokazałem w kolejnym fragmencie kodu.

```
if (distanceFromTop < 0) {
  // Jeżeli piłeczka uderzy w pobliżu górnej krawędzi paletki, należy zmniejszyć wartość ySpeed
  console.log("Uderzenie w pobliżu górnej krawędzi!");
  ySpeed -= 0.5;
} else if (distanceFromBottom < 0) {
  // Jeżeli piłeczka uderzy w pobliżu dolnej krawędzi paletki, należy zwiększyć wartość ySpeed
  console.log("Uderzenie w pobliżu dolnej krawędzi!");
  ySpeed += 0.5;
}
```

W ten sposób otrzymujemy dodatkowe powiadomienie, że piłeczka uderzyła w pobliżu górnej bądź dolnej krawędzi paletki, a wartość `ySpeed` została zmodyfikowana.

Jednak nie należy przesadzać z ilością komunikatów. Zachowaj ostrożność podczas dodawania komunikatów do gry, ponieważ ich zbyt duża ilość może bardzo szybko wprowadzić dezorientację, utrudnić ich odczytywanie, a także prowadzić do problemów związanych z wydajnością działania. Jeżeli dodasz generowanie komunikatu do funkcji np. `checkCollision`, wówczas w trakcie każdej klatki gry zostanie wyświetlony nowy komunikat. Najlepiej będzie ograniczyć komunikaty do określonych sytuacji, które nie zawsze występują, np. informować jedynie o kolizji, jak to ma miejsce w omawianym przykładzie.

Punktacja

Gra zwykle dostarcza więcej zabawy, gdy można ją wygrać lub przegrać. W grze Pong punkty zdobywało się za trafienie ściany znajdującej się za paletką przeciwnika. W takim przypadku położenie piłeczki i jej szybkość były zerowane do wartości początkowych i rozpoczynała się następna runda. W naszej grze również zastosujemy takie rozwiązanie. Jednak najpierw, aby przechowywać informacje o punktacji, konieczne jest utworzenie nowych zmiennych. Plik *script.js* uaktualnij do postaci przedstawionej na listingu 10.15.

Listing 10.15. Zmienne przeznaczone do przechowywania punktacji

```
--cięcie--
let leftPaddleTop = 10;
let rightPaddleTop = 30;

let leftScore = 0;
let rightScore = 0;

document.addEventListener("mousemove", e => {
  rightPaddleTop = e.y - canvas.offsetTop;
});
--cięcie--
```

W tym kodzie zostały zadeklarowane dwie nowe zmienne, `leftScore` i `rightScore`, a ich wartości początkowe wynoszą `0`. Później dodamy logikę przeznaczoną do inkrementacji wartości tych zmiennych po zdobyciu punktu przez gracza.

Następnym krokiem jest dodanie kodu, który będzie wyświetlał punktację. Ten kod powinien się znaleźć na końcu funkcji `draw`. Uaktualnij jej kod zgodnie z zawartością listingu 10.16.

Listing 10.16. Wyświetlenie punktacji

```
--cięcie--
  ctx.fillRect(
    width - PADDLE_WIDTH - PADDLE_OFFSET,
    rightPaddleTop,
    PADDLE_WIDTH,
    PADDLE_HEIGHT
  );

  // Wyświetlenie punktacji
  ctx.font = "30px monospace";
  ctx.textAlign = "left";
  ctx.fillText(leftScore.toString(), 50, 50);
  ctx.textAlign = "right";
  ctx.fillText(rightScore.toString(), width - 50, 50);
}

function update() {
--cięcie--
```

Ten kod używa pewnych nowych właściwości i metod płótna, które jeszcze nie były przedstawione w książce. Na początku właściwość `ctx.font` służy do zdefiniowania czcionki, która zostanie użyta podczas wyświetlania punktacji na ekranie. To jest podobne do deklaracji czcionki w CSS. W omawianym przykładzie czcionka ma wielkość 30 pikseli i styl *monospace*, co oznacza stałą szerokość znaków. Tego rodzaju czcionka jest stosowana zwykle w listingach kodu źródłowego, takich jak w tej książce. Oto przykład użycia `takiej czcionki`. Istnieje wiele czcionek o stałej szerokości znaków, ale ponieważ różne systemy operacyjne mogą mieć odmienne czcionki, w kodzie wskazaliśmy jedynie ogólne użycie tego stylu czcionki (`monospace`), pozostawiając systemowi operacyjnemu wybór właściwej. W większości systemów operacyjnych domyślną czcionką o stałej szerokości znaków jest Courier lub Courier New.

Następnie mamy właściwość `ctx.textAlign` użytą w celu zdefiniowania sposobu wyrównania tekstu. Zdecydowałem się na wyrównanie do lewej strony (`"left"`), ale ponieważ ma być ono zastosowane jedynie dla punktacji wyświetlanej po lewej stronie, przed wyświetleniem punktacji po prawej stronie wyrównanie zostało zmienione na `"right"`. W ten sposób jeśli punktacja stanie się dwucyfrowa, wówczas liczby zostaną przesunięte w kierunku środka ekranu, co zachowa wizualną równowagę.

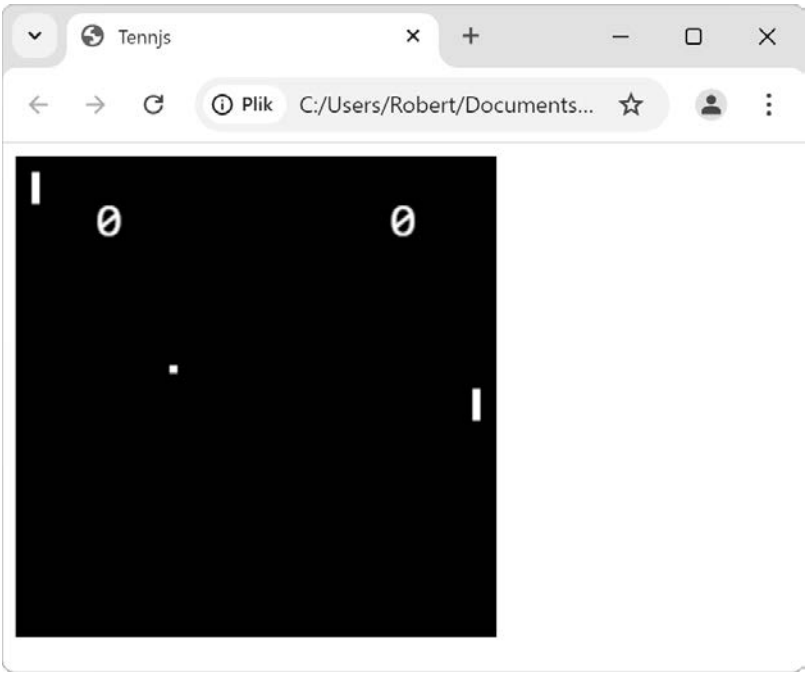
Aby wyświetlić punktację po lewej stronie, została użyta metoda `ctx.fillText`. Wymaga ona podania trzech parametrów: tekstu przeznaczonego do wyświetlenia oraz jego współrzędnych X i Y. Pierwszym parametrem musi być ciąg tekstowy, stąd wywołanie metody `toString` dla `leftScore`, aby przeprowadzić konwersję wartości liczbowej na postać ciągu tekstowego. Jako współrzędne X i Y została podana wartość `50`, więc tekst zostanie wyświetlony w pobliżu lewego górnego rogu płótna.

Uwaga

Znaczenie parametru współrzędnej X dla metody `fillText` zależy od wyrównania tekstu. W przypadku wyrównania do lewej strony ta wartość określa lewą krawędź tekstu, natomiast w przypadku wyrównania do prawej strony oznacza prawą krawędź tekstu.

Punktacja wyświetlana po prawej stronie jest obsługiwana w podobny sposób: definiujemy wyrównanie tekstu, a następnie wywołujemy metodę `fillText` w celu wyświetlenia punktacji. Tym razem wartością współrzędnej X jest `width - 50`, więc punktacja jest wyświetlana w takiej samej odległości od prawej krawędzi, w jakiej lewa punktacja znajduje się od lewej krawędzi.

Po odświeżeniu strony *index.html* zobaczysz wyświetloną punktację początkową, jak pokazałem na rysunku 10.6.



Rysunek 10.6. Wyświetlenie punktacji

Teraz trzeba zająć się obsługą sytuacji, w której piłeczka uderza w ściany boczne. Zamiast piłeczka odbić się od ściany, powinna zostać zwiększona punktacja odpowiedniego gracza, a położenie i szybkość piłeczki powinny zostać przywrócone do pierwotnych wartości. Rozpoczynamy od kolejnej refaktoryzacji i utworzenia funkcji, która będzie zerowała piłeczkę. To wymaga również pewnych zmian związanych z obsługą zmiennych przechowujących położenie piłeczki i jej szybkość. Zmiany niezbędne do wprowadzenia przedstawiłem na listingu 10.17.

Listing 10.17. Funkcja initBall

```
--cięcie--
const BALL_SIZE = 5;
let ballPosition;    ❶

let xSpeed;
let ySpeed;

function initBall() {
  ballPosition = { x: 20, y: 30 };    ❷
  xSpeed = 4;
  ySpeed = 2;
}

const PADDLE_WIDTH = 5;
--cięcie--
```

W kodzie oddzieliliśmy *deklaracje* zmiennych stanu piłeczki (`ballPosition`, `xSpeed` i `ySpeed`) od *inicjalizacji* tych zmiennych. Na przykład zmienna `ballPosition` została zadeklarowana na najwyższym poziomie programu ❶, ale zainicjalizowana w nowej funkcji `initBall` ❷, której nazwa jest skrótem dla *initialize ball* (inicjalizacja piłeczki). To samo dotyczy zmiennych `xSpeed` i `ySpeed`. Dzięki temu piłeczkę można wyzerować (jej położenie i szybkość) w dowolnym momencie — wystarczy wywołanie funkcji `initBall` i nie trzeba kopiować ani wklejać w różnych miejscach programu wartości zmiennych stanu piłeczki. Ta funkcja jest wywoływana przede wszystkim na początku programu, aby przygotować piłeczkę po raz pierwszy, a także za każdym razem, gdy piłeczka trafi w lewą bądź prawą ścianę, ponieważ wtedy następuje wyzerowanie piłeczki do stanu początkowego.

Zwróć uwagę na to, że nie można zadeklarować *ani* zainicjalizować zmiennych stanu piłeczki w funkcji `initBall`, np. za pomocą polecenia `let ballPosition = { x: 20, y: 30 }` w funkcji, ponieważ słowo kluczowe `let` definiuje nową zmienną *w bieżącym zasięgu*, którym w takim przypadku będzie funkcja `initBall`. Zatem zmienne byłyby dostępne jedynie w tej funkcji. Skoro oczekujemy, że te zmienne będą dostępne w całym programie, muszą być zadeklarowane poza jakąkolwiek funkcją. Ponieważ te zmienne będą inicjalizowane wielokrotnie, ich wartości są przypisywane w funkcji `initBall`, która może być wywoływana wiele razy.

Następnym krokiem jest modyfikacja kodu wykrywania kolizji w funkcji `checkCollision`, aby inkrementował punktację i zerował piłeczkę po trafieniu przez nią lewej bądź prawej ściany. Zmiany konieczne do wprowadzenia przedstawiłem na listingu 10.18.

Listing 10.18. Zdobycie punktu po trafieniu w ścianę boczną

```
  --cięcie--
    if (checkPaddleCollision(ball, rightPaddle)) {
      // Doszło do kolizji z prawą paletką
      let distanceFromTop = ball.top - rightPaddle.top;
      let distanceFromBottom = rightPaddle.bottom - ball.bottom;
      adjustAngle(distanceFromTop, distanceFromBottom);
      xSpeed = -Math.abs(xSpeed);
    }

    if (ball.left < 0) {    ❶
      rightScore++;
      initBall();
    }
    if (ball.right > width) {    ❷
      leftScore++;
      initBall();
    }

    if (ball.top < 0 || ball.bottom > height) {
      ySpeed = -ySpeed;
    }
}
  --cięcie--
```

Poprzednio sprawdzenie kolizji z lewą i prawą ścianą odbywało się za pomocą pojedynczego polecenia `if`, które powodowało odbicie piłeczki. Jednak teraz to sprawdzenie trzeba przeprowadzić oddzielnie dla ścian lewej i prawej, ponieważ punkty otrzymują różni gracze, w zależności od tego, która ściana została trafiona przez piłeczkę. Dlatego też wcześniejsze polecenie `if` zostało podzielone na dwa. Jeżeli piłeczka trafi w lewą ścianę ❶, wówczas inkrementowana jest wartość `rightScore`, zaś piłeczka jest zerowana za pomocą funkcji `initBall`. Natomiast w przypadku trafienia w prawą ścianę ❷ inkrementowana jest wartość `leftScore`, a piłeczka jest zerowana. Logika obsługująca kolizje piłeczki z górną i dolną ścianą pozostaje taka sama.

Skoro inicjalizacja zmiennych stanu piłeczki została przeniesiona do funkcji `initBall`, konieczne jest wywołanie tej funkcji przed uruchomieniem pętli gry, aby przygotować piłeczkę przed pierwszym użyciem. Przewiń w dół zawartość pliku *script.js* i uaktualnij kod przedstawiony na listingu 10.19 poprzez dodanie wywołania funkcji `initBall` przed wywołaniem `gameLoop`.

Listing 10.19. Pierwsze wywołanie funkcji initBall

```
--cięcie--
function gameLoop() {
  draw();
  update();
  checkCollision();

  // Ponowne wywołanie tej funkcji po upływie określonego czasu
  setTimeout(gameLoop, 30);
}

initBall();
gameLoop();
```

Teraz, po odświeżeniu w przeglądarce internetowej strony *index.html*, zobaczysz, że punktacja jest uaktualniana, gdy piłeczka uderzy w lewą bądź prawą ścianę. Ponadto po uderzeniu w ścianę następuje przywrócenie początkowego położenia i szybkości piłeczki. Oczywiście pokonanie komputera nie sprawia obecnie żadnych trudności, ponieważ nie porusza on jeszcze paletką.

Gracz sterowany przez komputer

Wprowadzimy teraz zmiany, dzięki którym gra stanie się znacznie bardziej wymagająca. Oczekujemy, że przeciwnik sterowany przez komputer będzie poruszał lewą paletką i będzie próbował zdobywać punkty. To można zaimplementować na różne sposoby. Jednak najprostsze podejście polega na tym, że komputer zawsze próbuje dopasować położenie paletki do aktualnego położenia piłeczki. Logika dla przeciwnika sterowanego przez komputer będzie bardzo prosta:

- Jeżeli górna krawędź piłeczki znajduje się ponad górną krawędzią paletki, wówczas paletkę trzeba przesunąć w górę.
- Jeżeli dolna krawędź piłeczki znajduje się poniżej dolnej krawędzi paletki, wówczas paletkę trzeba przesunąć w dół.
- W przeciwnym razie nie należy podejmować żadnych działań.

W przypadku takiego podejścia, jeśli komputer będzie mógł poruszać paletką z dowolną szybkością, nigdy nie spudłuje. Jednak taka sytuacja nie zapewni dobrej rozgrywki graczowi człowiekowi sterującemu prawą paletką. Dlatego też trzeba ograniczyć szybkość działania komputera. Na listingu 10.20 pokazałem, jak można to zrobić.

Listing 10.20. Ograniczenie szybkości poruszania paletką gracza sterowanego przez komputer

```
let canvas = document.querySelector("#canvas");
let ctx = canvas.getContext("2d");

let width = canvas.width;
let height = canvas.height;

const MAX_COMPUTER_SPEED = 2;

const BALL_SIZE = 5;
--cięcie--
```

Ograniczenie szybkości działania komputera odbyło się za pomocą zmiennej `MAX_COMPUTER_SPEED`. Przypisanie jej wartości `2` oznacza, że w trakcie jednej klatki gry komputer może przesunąć paletkę o maksymalnie dwa piksele.

Następnie zdefiniujemy funkcję o nazwie `followBall`, która będzie stosowała bardzo prostą postać sztucznej inteligencji odpowiedzialnej za poruszanie paletki gracza sterowanego przez komputer. Kod tej nowej funkcji zamieściłem na listingu 10.21. Ten kod umieść między funkcjami `draw` i `update`.

Listing 10.21. Kod funkcji obsługującej paletkę gracza sterowanego przez komputer

```
--cięcie--
function followBall() {
  let ball = {    ❶
    top: ballPosition.y,
    bottom: ballPosition.y + BALL_SIZE
  };
  let leftPaddle = {    ❷
    top: leftPaddleTop,
    bottom: leftPaddleTop + PADDLE_HEIGHT
  };

  if (ball.top < leftPaddle.top) {    ❸
    leftPaddleTop -= MAX_COMPUTER_SPEED;
  } else if (ball.bottom > leftPaddle.bottom) {    ❹
    leftPaddleTop += MAX_COMPUTER_SPEED;
  }
```

```
  }

  function update() {
    ballPosition.x += xSpeed;
    ballPosition.y += ySpeed;
    followBall();    ❺
}
  --cięcie--
```

W funkcji `followBall` definiujemy obiekty przedstawiające piłeczkę ❶ i lewą paletkę ❷. Każdy z nich ma właściwości `top` i `bottom` przedstawiające górną i dolną krawędź. Następnie za pomocą dwóch poleceń `if` implementujemy logikę poruszania paletką. Jeżeli górna krawędź piłeczki znajduje się ponad górną krawędzią paletki ❸, wówczas paletka jest przesuwana w górę poprzez odjęcie wartości `MAX_COMPUTER_SPEED` od `leftPaddleTop`. Analogicznie, jeżeli dolna krawędź piłeczki znajduje się poniżej dolnej krawędzi paletki ❹, wówczas paletka jest przesuwana w dół poprzez dodanie wartości `MAX_COMPUTER_SPEED` do `leftPaddleTop`.

Nowa funkcja `followBall` jest wywoływana w funkcji `update` ❺. W ten sposób poruszanie lewą paletką staje się częścią procesu uaktualnienia stanu gry, co odbywa się w trakcie każdej iteracji pętli gry.

Odśwież stronę w przeglądarce internetowej i sprawdź, czy jesteś w stanie pokonać komputer.

Koniec gry

Ostatnim etapem podczas tworzenia naszej gry jest umożliwienie jej wygrania (lub przegrania). To wymaga dołączenia pewnego rodzaju warunku kończącego grę, który spowoduje zatrzymanie pętli gry. W omawianym przykładzie pętla gry zakończy działanie, gdy którykolwiek z graczy zdobędzie 10 punktów. Na ekranie pojawi się wówczas komunikat „KONIEC GRY”.

Przede wszystkim trzeba zadeklarować zmienną przeznaczoną do śledzenia, czy gra jest zakończona. Będziemy używać tej zmiennej podczas podejmowania decyzji o kontynuowaniu działania funkcji `gameLoop`. Zmiany konieczne do wprowadzenia przedstawiłem na listingu 10.22.

Listing 10.22. Dodanie zmiennej gameOver

```
  --cięcie--
  let leftScore = 0;
  let rightScore = 0;
  let gameOver = false;    ❶

  document.addEventListener("mousemove", e => {
  --cięcie--

  function checkCollision() {
```

```
--cięcie--
  if (ball.right > width) {
    leftScore++;
    initBall();
  }
  if (leftScore > 9 || rightScore > 9) {    ❷
    gameOver = true;
  }
  if (ball.top < 0 || ball.bottom > height) {
    ySpeed = -ySpeed;
  }
}
--cięcie--
```

Gdzieś na początku pliku *script.js* trzeba zadeklarować zmienną o nazwie `gameOver`, która będzie wskazywała, czy gra się zakończyła ❶. Ta zmienna jest inicjalizowana z wartością `false`, aby gra nie zakończyła się jeszcze, zanim się rozpoczęła. Następnie w funkcji `checkCollision` sprawdzamy, czy którykolwiek z graczy zdobył powyżej 9 punktów ❷. Jeżeli tak, wartością zmiennej `gameOver` staje się `true`. Operację sprawdzenia można przeprowadzić gdziekolwiek, ale zdecydowałem się na funkcję `checkCollision`, aby w jednym miejscu umieścić logikę odpowiedzialną za sprawdzenie wyniku i jego inkrementację.

Następnym krokiem jest zdefiniowanie funkcji wyświetlającej komunikat „KONIEC GRY". Ponadto trzeba zmodyfikować pętlę gry, aby kończyła się po przypisaniu wartości `true` zmiennej `gameOver`. Zmiany konieczne do wprowadzenia przedstawiłem na listingu 10.23.

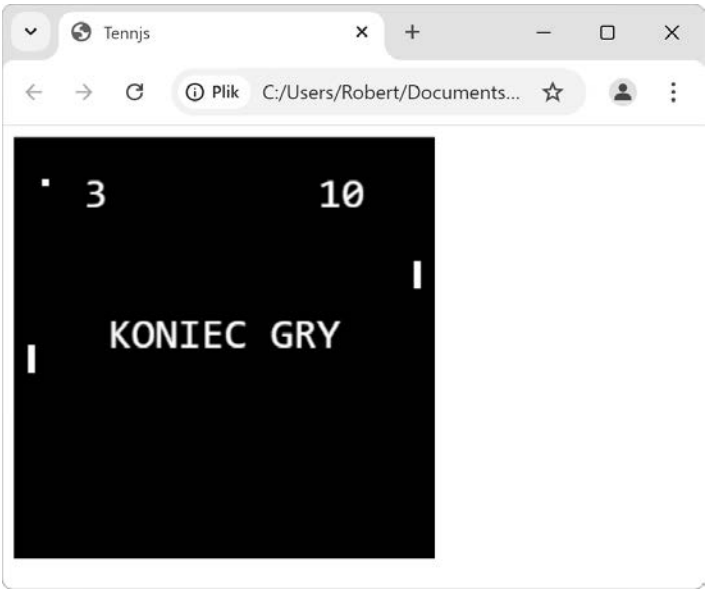
Listing 10.23. Zdefiniowanie logiki umożliwiającej zakończenie gry

```
--cięcie--
    if (ball.top < 0 || ball.bottom > height) {
      ySpeed = -ySpeed;
    }
  }

  function drawGameOver() {    ❶
    ctx.fillStyle = "white";
    ctx.font = "30px monospace";
    ctx.textAlign = "center";
    ctx.fillText("KONIEC GRY", width / 2, height / 2);
  }

  function gameLoop() {
    draw();
    update();
    checkCollision();
    if (gameOver) {    ❷
      draw();
      drawGameOver();
    } else {    ❸
      // Ponowne wywołanie tej funkcji po upływie określonego czasu
      setTimeout(gameLoop, 30);
    }
  }
```

Funkcję `drawGameOver` zdefiniowaliśmy po `checkCollision` ❶. Jej działanie polega na wyświetleniu komunikatu „KONIEC GRY” na środku płótna, z użyciem dużej czcionki w kolorze białym. Aby umieścić tekst na środku płótna, wyrównanie tekstu zostało zdefiniowane jako `"center"`. Ponadto dla współrzędnych X i Y komunikatu użyliśmy połowy szerokości i wysokości płótna. (W przypadku wyśrodkowania tekstu współrzędna X odwołuje się do środkowego w poziomie punktu tekstu).

W funkcji `gameLoop` znajduje się wywołanie `setTimeout` opakowane poleceniem warunkowym, które sprawdza wartość zmiennej `gameOver`. Jeżeli to `true` ❷, wówczas gra została zakończona, więc wywołane zostaną funkcje `draw` i `drawGameOver`. (Funkcja `draw` jest niezbędna do wyświetlenia ostatecznego wyniku, w przeciwnym razie wygrywający wciąż będzie widział zdobyte dotychczas 9 punktów). Jeżeli wartością `gameOver` jest `false` ❸, wówczas gra jest kontynuowana: pętla jest wykonywana jak wcześniej, poprzez zdefiniowanie `setTimeout` do wywołania `gameLoop` po upływie 30 ms.

Gdy zmienna `gameOver` będzie miała wartość `true`, a pętla gry nie będzie dłużej wykonywana, będzie to oznaczać praktyczne zakończenie rozgrywki. Po komunikacie „KONIEC GRY” na ekranie nic więcej nie zostanie wyświetlone — przynajmniej do chwili odświeżenia strony, co spowoduje ponowne rozpoczęcie wykonywania programu. Śmiało, wypróbuj to teraz: odśwież stronę *index.html* i sprawdź, czy jesteś w stanie pokonać komputer. Gdy którykolwiek z graczy zdobędzie więcej niż 9 punktów, na płótnie zostanie wyświetlony komunikat „KONIEC GRY”, jak pokazałem na rysunku 10.7.



Rysunek 10.7. Koniec gry

Mam nadzieję, że uda Ci się pokonać komputer, ale nie przejmuj się, jeśli tak nie będzie — gra jest dość trudna. Oto kilka podpowiedzi, dzięki którym możesz ją sobie ułatwić.

- Wydłuż czas między wyświetlaniem kolejnych klatek, co możesz zrobić w funkcji `gameLoop`.
- Wydłuż paletki.
- Zmniejsz maksymalną szybkość, z jaką może działać gracz sterowany przez komputer.
- Ułatw trafienie w krawędź paletki.
- Zwiększ wartość `ySpeed` po trafieniu krawędzi paletki.

Skoro masz w pełni działającą grę, możesz w niej wprowadzić dowolne zmiany. Jeżeli świetnie sobie radzisz w Pong, zbudowaną tutaj grę możesz nieco utrudnić. W znajdujących się poniżej ćwiczeniach zaproponowałem kilka sugestii. Możesz również spróbować dostosować do własnych potrzeb wygląd gry bądź zmienić wielkość płótna — teraz to jest Twoja gra.

WYPRÓBUJ SAMODZIELNIE

10.3. Zwiększ szybkość gry wraz ze zdobywaniem kolejnych punktów przez graczy (możesz to zrobić przez zwiększenie wartości `xSpeed` i `ySpeed` piłeczki bądź przez zmniejszenie wartości `setTimeout` w funkcji `gameLoop`).

10.4. Spowolnij paletkę, którą porusza gracz — to będzie wymagało mechanizmu podobnego do zastosowanego podczas obsługi ruchu paletki gracza sterowanego przez komputer. Trzeba zdefiniować pewną wartość maksymalną, o jaką może się przesunąć paletka gracza.

10.5. Dodaj drugą, wolniej poruszającą się piłeczkę.

Pełny kod źródłowy

Dla wygody na listingu 10.24 zamieściłem pełny kod pliku *script.js*.

Listing 10.24. Pełny kod źródłowy gry utworzonej w rozdziale

```
let canvas = document.querySelector("#canvas");
let ctx = canvas.getContext("2d");
let width = canvas.width;
let height = canvas.height;

const MAX_COMPUTER_SPEED = 2;

const BALL_SIZE = 5;
let ballPosition;
```

```
let xSpeed;
let ySpeed;

function initBall() {
  ballPosition = { x: 20, y: 30 };
  xSpeed = 4;
  ySpeed = 2;
}

const PADDLE_WIDTH = 5;
const PADDLE_HEIGHT = 20;
const PADDLE_OFFSET = 10;

let leftPaddleTop = 10;
let rightPaddleTop = 30;

let leftScore = 0;
let rightScore = 0;
let gameOver = false;

document.addEventListener("mousemove", e => {
  rightPaddleTop = e.y - canvas.offsetTop;
});

function draw() {
  // Wypełnienie płótna kolorem czarnym
  ctx.fillStyle = "black";
  ctx.fillRect(0, 0, width, height);

  // Wszystko pozostałe będzie w kolorze białym
  ctx.fillStyle = "white";

  // Narysowanie piłeczki
  ctx.fillRect(ballPosition.x, ballPosition.y, BALL_SIZE, BALL_SIZE);

  // Narysowanie paletek
  ctx.fillRect(
    PADDLE_OFFSET,
    leftPaddleTop,
    PADDLE_WIDTH,
    PADDLE_HEIGHT
  );
  ctx.fillRect(
    width - PADDLE_WIDTH - PADDLE_OFFSET,
    rightPaddleTop,
    PADDLE_WIDTH,
    PADDLE_HEIGHT
  );

  // Wyświetlenie punktacji
  ctx.font = "30px monospace";
  ctx.textAlign = "left";
  ctx.fillText(leftScore.toString(), 50, 50);
  ctx.textAlign = "right";
  ctx.fillText(rightScore.toString(), width - 50, 50);
}
```

```
function followBall() {
  let ball = {
    top: ballPosition.y,
    bottom: ballPosition.y + BALL_SIZE
  };
  let leftPaddle = {
    top: leftPaddleTop,
    bottom: leftPaddleTop + PADDLE_HEIGHT
  };

  if (ball.top < leftPaddle.top) {
    leftPaddleTop -= MAX_COMPUTER_SPEED;
  } else if (ball.bottom > leftPaddle.bottom) {
    leftPaddleTop += MAX_COMPUTER_SPEED;
  }
}

function update() {
  ballPosition.x += xSpeed;
  ballPosition.y += ySpeed;
  followBall();
}

function checkPaddleCollision(ball, paddle) {
  // Sprawdzenie, czy paletka i piłeczka nachodzą na siebie w pionie oraz w poziomie
  return (
    ball.left   < paddle.right &&
    ball.right  > paddle.left &&
    ball.top    < paddle.bottom &&
    ball.bottom > paddle.top
  );
}

function adjustAngle(distanceFromTop, distanceFromBottom) {
  if (distanceFromTop < 0) {
    // Jeżeli piłeczka uderzy w pobliżu górnej krawędzi paletki, należy zmniejszyć wartość ySpeed
    ySpeed -= 0.5;
  } else if (distanceFromBottom < 0) {
    // Jeżeli piłeczka uderzy w pobliżu dolnej krawędzi paletki, należy zwiększyć wartość ySpeed
    ySpeed += 0.5;
  }
}

function checkCollision() {
  let ball = {
    left: ballPosition.x,
    right: ballPosition.x + BALL_SIZE,
    top: ballPosition.y,
    bottom: ballPosition.y + BALL_SIZE
  }

  let leftPaddle = {
    left: PADDLE_OFFSET,
    right: PADDLE_OFFSET + PADDLE_WIDTH,
    top: leftPaddleTop,
    bottom: leftPaddleTop + PADDLE_HEIGHT
```

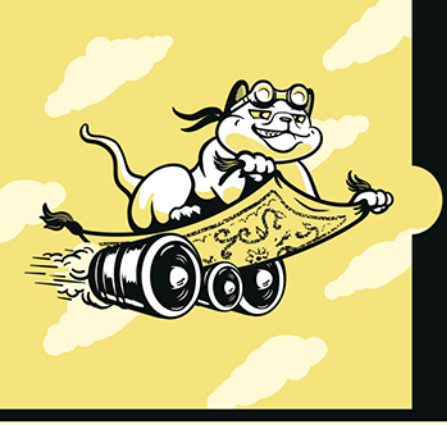
```
  };

  let rightPaddle = {
    left: width - PADDLE_WIDTH - PADDLE_OFFSET,
    right: width - PADDLE_OFFSET,
    top: rightPaddleTop,
    bottom: rightPaddleTop + PADDLE_HEIGHT
  };

  if (checkPaddleCollision(ball, leftPaddle)) {
    // Doszło do kolizji z lewą paletką
    let distanceFromTop = ball.top - leftPaddle.top;
    let distanceFromBottom = leftPaddle.bottom - ball.bottom;
    adjustAngle(distanceFromTop, distanceFromBottom);
    xSpeed = Math.abs(xSpeed);
  }

  if (checkPaddleCollision(ball, rightPaddle)) {
    // Doszło do kolizji z prawą paletką
    let distanceFromTop = ball.top - rightPaddle.top;
    let distanceFromBottom = rightPaddle.bottom - ball.bottom;
    adjustAngle(distanceFromTop, distanceFromBottom);
    xSpeed = -Math.abs(xSpeed);
  }

  if (ball.left < 0) {
    rightScore++;
    initBall();
  }
  if (ball.right > width) {
    leftScore++;
    initBall();
  }
  if (leftScore > 9 || rightScore > 9) {
    gameOver = true;
  }
  if (ball.top < 0 || ball.bottom > height) {
    ySpeed = -ySpeed;
  }
}

function drawGameOver() {
  ctx.fillStyle = "white";
  ctx.font = "30px monospace";
  ctx.textAlign = "center";
  ctx.fillText("KONIEC GRY", width / 2, height / 2);
}

function gameLoop() {
  draw();
  update();
  checkCollision();

  if (gameOver) {
    draw();
    drawGameOver();
```

```
  } else {
    // Ponowne wywołanie tej funkcji po upływie określonego czasu
    setTimeout(gameLoop, 30);
  }
}

initBall();
gameLoop();
```

Podsumowanie

W rozdziale zupełnie od początku utworzyliśmy grę w JavaScripcie. Podstawowe mechanizmy zastosowane w grze — pętla gry, wykrywanie kolizji i generowanie elementów — są powszechnie używane. Zatem zdobyta tutaj wiedza pozwala rozpocząć samodzielne tworzenie wszelkiego rodzaju gier dwuwymiarowych. Na przykład możesz spróbować samodzielnie zaimplementować grę typu Breakout lub Snake. Jeżeli potrzebujesz pomocy z logiką używaną w takich grach, w internecie znajdziesz wiele samouczków. Miłej zabawy!


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



JAVASCRIPT: ŁATWY START, WIELKIE MOŻLIWOŚCI!

JavaScript jest niewielkim językiem skryptowym o imponujących możliwościach. Niegdyś był kojarzony głównie z efektami specjalnymi na wczesnych stronach internetowych. Dziś dzięki ciągłemu rozwojowi i bogatemu ekosystemowi jest jednym z najważniejszych języków w świecie programowania, idealnym do rozpoczęcia przygody z tworzeniem kodu. Wszystko, czego potrzebujesz do szybkiej nauki programowania w JavaScripcie, to trochę chęci, przeglądarka internetowa i ta książka!

To interesujące, zwięzłe i wyjątkowo praktyczne wprowadzenie do programowania w języku JavaScript. Już podczas lektury pierwszych stron zaczniesz pisać własny kod, będziesz znajdować rozwiązania różnych wyzwań, a także tworzyć aplikacje internetowe i zabawne gry. Rozpoczniesz od poznania podstawowych koncepcji stosowanych w programowaniu, takich jak zmienne, tablice, obiekty, funkcje, konstrukcje warunkowe, pętle itd. Następnie nauczysz się łączyć skrypty JavaScript z kodem HTML i CSS, aby tworzyć interaktywne aplikacje internetowe. Ze swoich nowych umiejętności skorzystasz podczas pracy nad trzema większymi projektami: grą w stylu *Pong*, aplikacją generującą muzykę i platformą przeznaczoną do wizualizacji danych pobranych za pomocą API.

Błyskawicznie nauczysz się:

- uaktualniać strony internetowe w czasie rzeczywistym
- wywoływać funkcje w reakcji na zdarzenia
- generować grafikę i animacje za pomocą JavaScriptu
- wizualizować dane za pomocą biblioteki D3.js i grafiki w formacie SVG
- komponować muzykę elektroniczną za pomocą biblioteki Tone.js i API Web Audio

Nick Morgan jest inżynierem oprogramowania w Airbnb, gdzie zajmuje się podstawowymi usługami umożliwiającymi działanie serwisu. Wcześniej pracował w Twitterze. Dorastał w Wielkiej Brytanii. Na Uniwersytecie Surrey uzyskał dyplom z muzyki i nagrań dźwiękowych. Obecnie mieszka w Colorado z żoną, dwiema córkami, trzema kotami i psem.

Helion

helion.pl

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

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-289-2067-5

9 788328 920675

Cena: 99,00 zł

no starch press