Tworzenie gier w języku HTML5 dla profesjonalistów

Wydanie II

Aditya Ravi Shankar

Helion

Apress

Tytuł oryginału: Pro HTML5 Games: Learn to Build your Own Games using HTML5 and JavaScript, 2nd Edition

Tłumaczenie: Joanna Zatorska

ISBN: 978-83-283-4325-2

HELION SA
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/twghp2.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/twghp2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Podstawy języka HTML5 i JavaScript

HTML5, czyli najnowsza wersja standardu HTML, udostępnia wiele nowych funkcjonalności, które usprawniają interakcje i zapewniają lepsze wsparcie dla multimediów. Dzięki nim (np. dzięki elementom `canvas`, `audio` i `video`) można tworzyć dość bogate i interaktywne aplikacje przeznaczone do uruchamiania w przeglądarce, które nie wymagają korzystania z zewnętrznych dodatków, takich jak Flash.

Chociaż standard HTML5 jest stale rozwijany i ulepszany, będąc nadal tzw. żywym standardem, wszystkie elementy potrzebne do tworzenia fantastycznych gier są powszechnie wspierane przez nowoczesne przeglądarki (Google Chrome, Mozilla Firefox, Internet Explorer 9+, Microsoft Edge, Safari i Opera).

W ciągu połowy ostatniej dekady (czyli od czasu, gdy napisałem pierwszą wersję tej książki) wsparcie dla HTML5 stało się standardem we wszystkich nowoczesnych przeglądarkach, zarówno na komputerach stacjonarnych, jak i na urządzeniach mobilnych. Oznacza to, że obecnie w języku HTML5 można tworzyć gry, które da się z łatwością dostosować do działania zarówno na urządzeniach mobilnych, jak i stacjonarnych, działających pod kontrolą różnych systemów operacyjnych.

Aby rozpocząć samodzielne tworzenie gier w HTML5, wystarczy dobry edytor tekstowy, w którym będziemy wpisywać kod (obecnie korzystam z programu Visual Studio Code w wersji na komputery Mac i PC — *https://code.visualstudio.com/*) oraz nowoczesna przeglądarka zgodna z HTML5 (korzystam przede wszystkim z Google Chrome). Po zainstalowaniu ulubionego edytora tekstowego oraz przeglądarki zgodnej z HTML5 można zacząć tworzenie swojej pierwszej strony HTML5.

Podstawowa strona HTML5

Struktura dokumentu HTML5 jest bardzo podobna do struktury dokumentu utworzonego w poprzednich wersjach języka. Różnica polega na tym, że w HTML5 znacznik `DOCTYPE` znajdujący się na początku dokumentu jest znacznie uproszczony. Na podstawie prostszego znacznika `DOCTYPE` przeglądarka podczas interpretacji dokumentu rozpoznaje konieczność użycia najnowszego standardu.

Listing 1.1 zawiera szkielet bardzo prostego pliku HTML5, który posłuży nam jako punkt wyjścia do pracy w nad zadaniami opisanymi dalej w tym rozdziale. Wykonanie tego kodu wymaga zapisania go w pliku HTML, a następnie otwarcia w przeglądarce internetowej. Jeśli wszystko zostanie zrobione poprawnie, w oknie przeglądarki ujrzymy napis „Witaj, świecie!”.

Listing 1.1. *Szkielet podstawowego pliku HTML5*

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <title>Sample HTML5 File</title>
        <script type="text/javascript">
            // Ta funkcja zostanie wywołana jeden raz, po całkowitym wczytaniu strony
            function pageLoaded(){
                alert("Witaj, Świecie!");
            }
        </script>
    </head>
    <body onload="pageLoaded();">

    </body>
</html>
```

■ **Uwaga:** aby wywołać funkcję `pageLoaded()`, korzystamy ze zdarzenia `onload` w elemencie `body`. Dzięki temu mamy pewność, że strona zostanie całkowicie wczytana, zanim zaczniemy wykonywać nasze zadania. Będzie to szczególnie istotne, gdy będziemy przetwarzać takie elementy jak obrazy i dźwięk. Próba dostępu do tych elementów przed ich wczytaniem w przeglądarce spowoduje powstanie błędów JavaScriptu lub będzie miała inne nieoczekiwane skutki.

Zanim zaczniemy tworzyć gry, musimy poznać podstawowe bloki, za pomocą których będziemy budować swoje gry. Oto najważniejsze elementy:

- `canvas`, który służy do renderowania kształtów i obrazów;
- `audio`, który służy do dodawania dźwięków i muzyki w tle;
- `image`, który służy do wczytywania grafiki gry oraz wyświetlania jej w elemencie `canvas`;
- funkcje zegara przeglądarki oraz pętle gry służące do obsługi animacji.

Element canvas

Najważniejszym elementem, z którego będziemy korzystać w naszych grach, jest nowy element `canvas`. Zgodnie ze specyfikacją standardu HTML5 „Element canvas oferuje skryptom kanwę rysunku w postaci bitmapy zależnej od rozdzielczości, na której można w locie renderować wykresy, grafikę gier, obrazy i inne elementy graficzne”. Kompletna specyfikacja jest dostępna pod adresem *https://html.spec.whatwg.org/multipage/scripting.html#the-canvas-element*.

Element `canvas` umożliwia rysowanie prostych kształtów, takich jak linie, koła i prostokąty, a także obrazów oraz tekstu. Ten element został zoptymalizowany pod kątem szybkiego rysowania. W przeglądarkach zaczęto oferować przyspieszanie GPU dla renderowania dwuwymiarowych obiektów znajdujących się w `canvas`, aby umożliwić szybkie działanie gier i animacji opartych na tym elemencie.

Korzystanie z `canvas` jest dość proste. Wystarczy umieścić znacznik `<canvas>` w elemencie `body` pliku HTML5, który wcześniej utworzyliśmy. Przykład zamieszczono w listingu 1.2.

Listing 1.2. *Tworzenie elementu canvas*

```
<body onload="pageLoaded();">
    <canvas width="640" height="480" id="testcanvas" style="border: 1px solid black;">
        Twoja przeglądarka nie obsługuje elementu canvas języka HTML5.
        ↪Wyświetl ten dokument w nowszej przeglądarce.
    </canvas>
</body>
```

Kod z listingu 1.2 utworzy element `canvas` o szerokości 640 pikseli i wysokości 480 pikseli. Sam element `canvas` wygląda jak pusty obszar (otoczony czarną ramką, zgodnie z podanym stylem). Korzystając z kodu JavaScriptu, możemy zacząć rysowanie na tym prostokątnym obszarze.

■ **Uwaga:** przeglądarki, które nie obsługują elementu `canvas`, zignorują znacznik `<canvas>` i wyrenderują wszystko, co znajduje się wewnątrz tego znacznika. Możemy to wykorzystać i zaprezentować użytkownikom starszych przeglądarek alternatywne treści lub komunikat, w którym poprosimy ich o wybranie nowocześniejszej przeglądarki.

W elemencie `canvas` rysujemy, korzystając z tzw. głównego kontekstu renderowania. Kontekst ten możemy uzyskać za pomocą metody `getContext()` obiektu `canvas`. Metoda `getContext()` przyjmuje jeden parametr: potrzebny nam typ kontekstu. W naszych grach będziemy stosować kontekst `2d`.

Listing 1.3 pokazuje, że dostęp do elementu `canvas` i jego kontekstu możemy uzyskać po wczytaniu strony, modyfikując metodę `pageLoaded()`.

Listing 1.3. *Dostęp do kontekstu obiektu canvas*

```
<script type="text/javascript">
    function pageLoaded(){

        // Pobieramy uchwyt do obiektu canvas
        var canvas = document.getElementById("testcanvas");

        // Pobieramy kontekst 2d tego obiektu canvas
        var context = canvas.getContext("2d");

        // Nasz kod rysowania...
    }
</script>
```

■ **Uwaga:** wszystkie przeglądarki obsługują kontekst `2d` potrzebny do rysowania grafiki dwuwymiarowej. Większość przeglądarek implementuje także inne typy kontekstu, np. `webgl` lub `experimental-webgl`, służące do tworzenia grafiki trójwymiarowej.

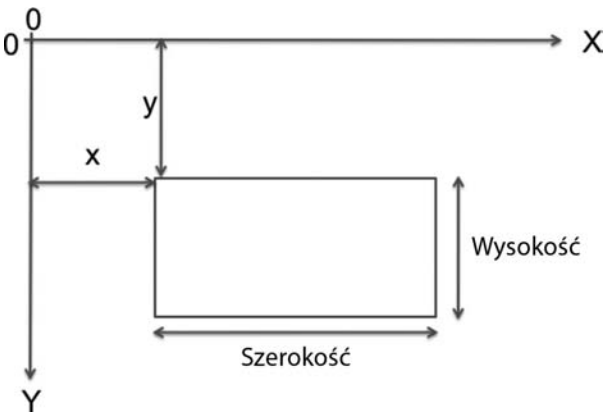
Wydaje się, że ten kod na razie niczego nie robi. Jednak mamy już dostęp do obiektu kontekstu `2d`. Udostępnia on ogromną liczbę metod, za pomocą których możemy rysować na ekranie elementy naszej gry. Te metody umożliwiają m.in.:

- rysowanie prostokątów,
- rysowanie złożonych ścieżek (linii, łuków itd.),
- rysowanie tekstu,
- dostosowywanie stylów rysowania (kolorów, przezroczystości, tekstur itd.),
- rysowanie obrazów,
- przekształcanie i obracanie.

Bliżej przyjrzymy się tym metodom w kolejnych podrozdziałach.

Rysowanie prostokątów

Zanim zaczniemy rysować w elemencie `canvas`, musimy zrozumieć zasady rządzące układem współrzędnych stosowanym w tym elemencie. Obiekt `canvas` wykorzystuje układ współrzędnych, którego początek (0, 0) znajduje się w górnym lewym rogu obiektu. Wartości x rosną w prawo, natomiast wartości y rosną w dół, jak przedstawiono na rysunku 1.1.



Rysunek 1.1. *Układ współrzędnych obiektu canvas*

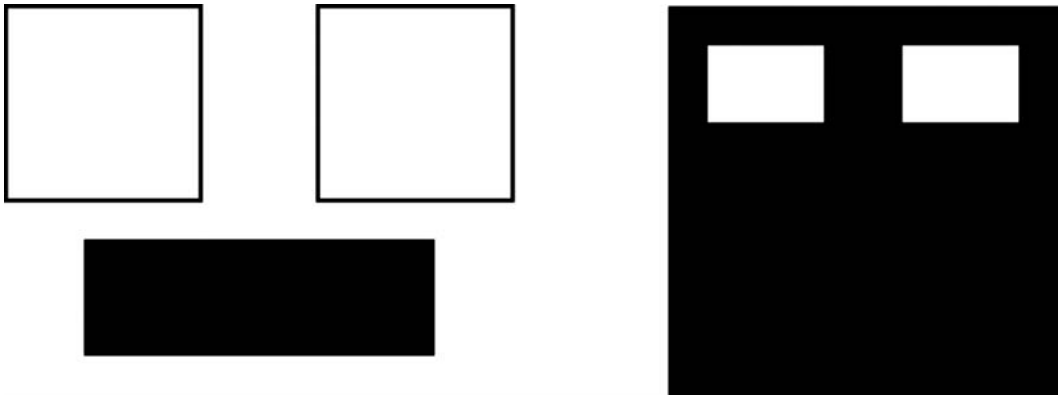
Prostokąt można narysować w obiekcie `canvas` za pomocą metod kontekstu dotyczących prostokątów:

- `fillRect(x, y, width, height)` — rysuje wypełniony prostokąt;
- `strokeRect(x, y, width, height)` — rysuje ramkę prostokątną;
- `clearRect(x, y, width, height)` — czyści określony prostokątny obszar i sprawia, że jest on całkowicie przezroczysty.

Listing 1.4. *Rysowanie prostokątów w obiekcie canvas*

```
// WYPEŁNIONE PROSTOKĄTY
// Rysowanie jednolitego prostokąta o szerokości i wysokości 100 pikseli w punkcie (200,10)
context.fillRect(200, 10, 100, 100);
// Rysowanie jednolitego prostokąta o szerokości 90 pikseli i wysokości 30 pikseli w punkcie (50,70)
context.fillRect(50, 70, 90, 30);

// OBRYSOWANE PROSTOKĄTY
// Rysowanie prostokątnej ramki o szerokości i wysokości 50 pikseli w punkcie (110, 10)
context.strokeRect(110, 10, 50, 50);
// Rysowanie prostokątnej ramki o szerokości i wysokości 50 pikseli w punkcie (30, 10)
context.strokeRect(30, 10, 50, 50);

// USUWANIE PROSTOKĄTNYCH OBSZARÓW
// Usuwanie prostokątnego obszaru o szerokości 30 pikseli i wysokości 20 w punkcie (210, 20)
context.clearRect(210, 20, 30, 20);
// Usuwanie prostokątnego obszaru o szerokości 30 pikseli i wysokości 20 w punkcie (260, 20)
context.clearRect(260, 20, 30, 20);
```

Za pomocą kodu z listingu 1.4 narysujemy kilka prostokątów w górnym lewym rogu elementu `canvas`, jak na rysunku 1.2. Aby to sprawdzić, należy umieścić ten kod w dolnej części metody `pageLoaded()`, zapisać plik i odświeżyć stronę w przeglądarce. Powinniśmy ujrzeć wynik wprowadzonych zmian.



Rysunek 1.2. *Rysowanie prostokątów w elemencie canvas*

Rysowanie ścieżek złożonych

Jeśli zwykłe prostokąty okażą się niewystarczające, możemy skorzystać z innych metod kontekstu, za pomocą których narysujemy złożone kształty:

- `beginPath()` — rozpoczyna rejestrowanie nowego kształtu;
- `closePath()` — zamyka ścieżkę, rysując linię od bieżącego punktu do punktu początkowego;
- `fill()`, `stroke()` — wypełnia lub obrysowuje narysowany kształt;
- `moveTo(x, y)` — przenosi punkt rysowania do punktu o współrzędnych x, y;
- `lineTo(x, y)` — rysuje linię z bieżącego punktu do punktu o współrzędnych x, y;
- `arc(x, y, radius, startAngle, endAngle, anticlockwise)` — rysuje łuk o podanym promieniu, w punkcie x, y.

Za pomocą tych metod możemy narysować ścieżkę złożoną, wykonując po kolei następujące kroki:

1. Za pomocą metody `beginPath()` rozpoczynamy rejestrowanie nowego kształtu.
2. Za pomocą metod `moveTo()`, `lineTo()` i `arc()` tworzymy kształt.
3. Opcjonalnie zamykamy kształt za pomocą metody `closePath()`.
4. Za pomocą metody `stroke()` lub `fill()` rysujemy obrys lub wypełniamy kształt. Jeśli użyjemy metody `fill()`, wszystkie otwarte ścieżki zostaną automatycznie zamknięte.

Kod z listingu 1.5 tworzy trójkąty, łuki i kształty widoczne na rysunku 1.3.

Listing 1.5. *Rysowanie kształtów złożonych w elemencie canvas*

```
// RYSOWANIE KSZTAŁTÓW ZŁOŻONYCH
// Rysowanie wypełnionego trójkąta
context.beginPath();
context.moveTo(10, 120);      // Zaczynamy rysowanie w punkcie 10, 120
context.lineTo(10, 180);
context.lineTo(110, 150);
context.fill();               // Zamykamy kształt i wypełniamy go

// Rysowanie obrysowanego trójkąta
context.beginPath();
```

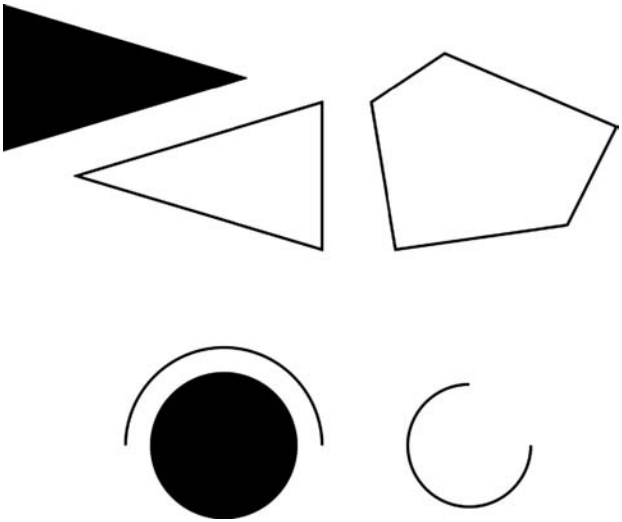
```
context.moveTo(140, 160); // Zaczynamy rysowanie w punkcie 140, 160
context.lineTo(140, 220);
context.lineTo(40, 190);
context.closePath();
context.stroke();

// Rysowanie bardziej złożonego zestawu linii
context.beginPath();
context.moveTo(160, 160); // Zaczynamy rysowanie w punkcie 160, 160
context.lineTo(170, 220);
context.lineTo(240, 210);
context.lineTo(260, 170);
context.lineTo(190, 140);
context.closePath();
context.stroke();

// RYSOWANIE ŁUKÓW I KÓŁ
// Rysowanie półkola
context.beginPath();
// Rysowanie łuku w punkcie (400, 50) o promieniu 40, od kąta 0 do kąta 180, w kierunku odwrotnym do wskazówek zegara
// PI radianów = 180 stopni
context.arc(100, 300, 40, 0, Math.PI, true);
context.stroke();

// Rysowanie pełnego koła
context.beginPath();
// Rysowanie łuku w punkcie (500, 50) o promieniu 30, od kąta 0 do kąta 360, w kierunku odwrotnym do wskazówek zegara
// 2*PI radianów = 360 stopni
context.arc(100, 300, 30, 0, 2 * Math.PI, true);
context.fill();

// Rysowanie trzech czwartych okręgu
context.beginPath();
// Rysowanie łuku w punkcie (400, 100) o promieniu 25, od kąta 0 do kąta 270, w kierunku zgodnym ze wskazówkami zegara
// (3/2*PI radianów = 270 stopni)
context.arc(200, 300, 25, 0, 3 / 2 * Math.PI, false);
context.stroke();
```

Rysowanie tekstu

Kontekst zawiera też metody służące do rysowania tekstu w elemencie `canvas`:

- `strokeText(text, x, y)` — rysuje obrys tekstu w punkcie (x, y);
- `fillText(text, x, y)` — wypełnia tekst znajdujący się w punkcie (x, y).

W przeciwieństwie do tekstu znajdującego się w innych elementach HTML tekst umieszczony w obiekcie `canvas` nie ma opcji układu CSS, takich jak zawijanie tekstu czy marginesy wewnętrzne i zewnętrzne. Jednakże wygląd tekstu można zmodyfikować, ustawiając krój czcionki stosowany w kontekście, obrys oraz wypełnienie (patrz listing 1.6).



Rysunek 1.3. *Rysowanie złożonych kształtów w elemencie canvas*

Listing 1.6. *Rysowanie tekstu w obiekcie canvas*

```
// RYSOWANIE TEKSTU
context.fillText("To jest jakiś tekst...", 330, 40);

// Modyfikowanie kroju czcionki
context.font = "10pt Arial";
context.fillText("To jest krój 10pt Arial...", 330, 60);

// Rysowanie obrysowanego tekstu
context.font = "16pt Arial";
context.strokeText("To jest obrysowany tekst o kroju 16pt Arial...", 330, 80);
```

Kod z listingu 1.6 narysuje tekst widoczny na rysunku 1.4.

Rysunek 1.4. *Rysowanie tekstu w obiekcie canvas*

Ustawiając właściwość `font`, możemy użyć dowolnej właściwości CSS dostępnej dla krojów czcionek. Jak widać na powyższym przykładzie, chociaż nie daje nam to tego samego poziomu elastyczności w formatowaniu, jakie zapewnia HTML i CSS, nadal możemy uzyskać różnorodne efekty, stosując metody tekstu oferowane przez obiekt `canvas`. Oczywiście, o wiele lepszy efekt uzyskamy, stosując kolory.

Dostosowywanie stylów rysowania (kolorów i tekstur)

Wszystko, co dotychczas narysowaliśmy, było czarne, ale tylko dlatego, że jest to domyślny kolor rysowania w obiekcie `canvas`. Mamy więcej możliwości. W obiekcie `canvas` możemy zastosować style i dopasować wygląd

linii, kształtów i tekstu. Możemy rysować różnymi kolorami, używać różnych stylów linii, przezroczystości, a nawet wypełniać kształty teksturami.

Jeśli chcemy zastosować kolory w kształcie, możemy skorzystać z dwóch ważnych właściwości:

- `fillStyle` — ustawia domyślny kolor wszystkich przyszłych operacji wypełniania;
- `strokeStyle` — ustawia domyślny kolor wszystkich przyszłych operacji obrysowywania.

Obydwie właściwości mogą przyjmować wartość w postaci poprawnego koloru CSS. Dotyczy to również wartości `rgb()` i `rgba()`, a także stałych wartości kolorów. Przykładowo: kod `context.fillStyle = "red";` zdefiniuje czerwony domyślny kolor, który będzie używany we wszystkich przyszłych operacjach wypełniania (`fillRect`, `fillText` i `fill`).

Ponadto metoda `createTexture()` obiektu `context` tworzy teksturę na podstawie obrazu, tekstura ta może posłużyć jako styl wypełnienia. Zanim będziemy mogli użyć obrazu, musimy go wczytać do przeglądarki. Na razie wystarczy dodać w naszym pliku HTML znacznik `<img>` za znacznikiem `<canvas>`:

```
<img src="fire.png" id="fire">
```

Kod z listingu 1.7 narysuje kolorowe i wypełnione teksturą prostokąty widoczne na rysunku 1.5.

Listing 1.7. *Rysowanie z wykorzystaniem kolorów i tekstur*

```
// STYLE WYPEŁNIENIA I KOLORY
// Ustawiamy kolor wypełnienia na czerwony
context.fillStyle = "red";
// Rysujemy prostokąt wypełniony czerwonym kolorem
context.fillRect(310, 160, 100, 50);

// Ustawiamy kolor obrysu na zielony
context.strokeStyle = "green";
// Rysujemy prostokąt z zielonym obrysem
context.strokeRect(310, 240, 100, 50);

// Ustawiamy kolor wypełnienia na żółty za pomocą metody rgb()
context.fillStyle = "rgb(255, 255, 0)";
// Rysujemy prostokąt wypełniony żółtym kolorem
context.fillRect(420, 160, 100, 50);

// Ustawiamy kolor wypełnienia na zielony z przezroczystością, czyli alpha o wartości 0.6
context.fillStyle = "rgba(0, 255, 0, 0.6)";
// Rysowanie półprzezroczystego prostokąta wypełnionego zielonym kolorem
context.fillRect(450, 180, 100, 50);

// TEKSTURY
// Pobieramy uchwyt do obiektu obrazu
var fireImage = document.getElementById("fire");
var pattern = context.createPattern(fireImage, "repeat");

// Ustawiamy styl wypełnienia na nowo utworzony wzór
context.fillStyle = pattern;
// Rysujemy prostokąt wypełniony wzorem
context.fillRect(420, 240, 130, 50);
```

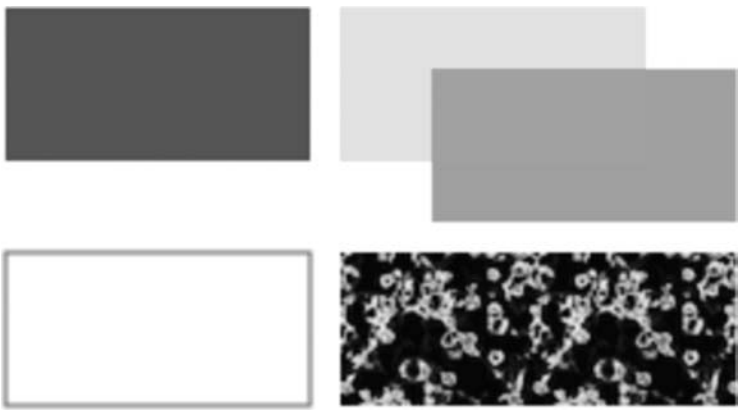
Oprócz powyższych metod obiekt `canvas` oferuje kilka innych, za pomocą których można podczas rysowania stosować gradienty kolorów, cienie i wzory. Zachęcam do poświęcenia większej ilości czasu na pogłębienie wiedzy o interfejsie API obiektów `canvas` i `context`.

Rysowanie obrazów

Chociaż za pomocą poznanych dotychczas metod rysowania możemy sporo osiągnąć, nadal nie wiemy, jak korzystać z obrazów. Gdy się dowiemy, jak rysować obrazy, będziemy mogli rysować tła gier, sprite'y postaci oraz efekty takie jak wybuchy, które ożywią naszą grę.

Obrazy i elementy typu sprite można rysować w obiekcie `canvas` za pomocą metody `drawImage()`. Obiekt `context` zapewnia trzy różne wersje tej metody:

- `drawImage(image, x, y)` — rysuje obraz w obiekcie `canvas` w punkcie (x, y);
- `drawImage(image, x, y, width, height)` — skaluje obraz do podanej szerokości i wysokości, a następnie rysuje go w punkcie (x, y);



Rysunek 1.5. *Rysowanie z wykorzystaniem kolorów i tekstur*

- `drawImage(image, sourceX, sourceY, sourceWidth, sourceHeight, x, y, width, height)` — wycina prostokąt z obrazu, rozpoczynając od punktu (`sourceX`, `sourceY`), o rozmiarach (`sourceWidth`, `sourceHeight`), skaluje go do podanej szerokości i wysokości, a potem rysuje w obiekcie `canvas` w punkcie (x, y).

Zanim przejdziemy do rysowania obrazów, musimy wczytać do przeglądarki kolejny obraz. Dodamy jeszcze w naszym pliku HTML jeden znacznik `<img>` za znacznikiem `<canvas>`:

```
<img src="spaceship.png" id="spaceship">
```

Po wczytaniu obrazu możemy go narysować, korzystając z kodu z listingu 1.8.

Listing 1.8. *Rysowanie obrazów*

```
// RYSOWANIE OBRAZÓW
// Pobieranie uchwytu do obrazu
var image = document.getElementById("spaceship");

// Rysowanie obrazu w punkcie (0, 350)
context.drawImage(image, 0, 350);

// Skalowanie obrazu do połowy oryginalnego rozmiaru
context.drawImage(image, 0, 400, 100, 25);

// Rysowanie fragmentu obrazu
context.drawImage(image, 0, 0, 60, 50, 0, 420, 60, 50);
```

Kod z listingu 1.8 narysuje obrazy przedstawione na rysunku 1.6. Ostatni przykład z listingu 1.8, w którym rysujemy tylko część obrazu, będzie szczególnie przydatny, gdy zaczniemy korzystać z arkuszy sprite'ów, które umożliwiają zebranie w jednym miejscu zasobów gry i przechowywanie wielu sprite'ów w jednym dużym obrazie.

Rysunek 1.6. *Rysowanie obrazów*

Przekształcanie i obracanie

Obiekt `context` oferuje kilka metod służących do przekształcania układu współrzędnych, na którym rysujemy elementy. Są to następujące metody:

- `translate(x, y)` — przesuwa obiekt `canvas` i jego początek do innego punktu (x, y);
- `rotate(angle)` — obraca obiekt `canvas` zgodnie z kierunkiem wskazówek zegara o określony kąt (radiany);
- `scale(x, y)` — skaluje narysowany obiekt poprzez przemnożenie odpowiednich współrzędnych przez wartości x i y.

Metody te zwykle wykorzystuje się do obracania rysowanych obiektów lub sprite'ów. W tym celu możemy:

- dokonać translacji punktu początkowego układu współrzędnych obiektu `canvas` do położenia obiektu;
- obrócić obiekt `canvas` o określony kąt;
- narysować obiekt;
- przywrócić obiekt `canvas` do stanu początkowego.

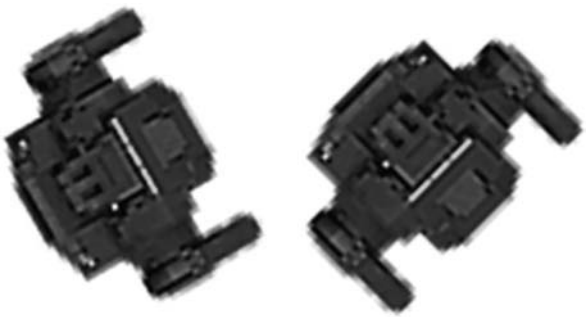
Sprawdźmy, jak obrócić obiekty przed ich narysowaniem. Odpowiedni kod zawiera listing 1.9.

Listing 1.9. *Obracanie obiektów przed ich narysowaniem*

```
// OBRACANIE I TRANSLACJA
// Translacja początku układu współrzędnych do miejsca położenia obiektu
context.translate(250, 370);
// Obrót nowego początku o 60 stopni
context.rotate(Math.PI / 3);
context.drawImage(image, 0, 0, 60, 50, -30, -25, 60, 50);
```

```
// Przywracanie do stanu początkowego poprzez odwrócenie obrotu i translacji
context.rotate(-Math.PI / 3);
context.translate(-240, -370);

// Translacja początku układu współrzędnych do miejsca położenia obiektu
context.translate(300, 370);
// Obracanie nowego początku
context.rotate(3 * Math.PI / 4);
context.drawImage(image, 0, 0, 60, 50, -30, -25, 60, 50);
// Przywracanie do stanu początkowego poprzez odwrócenie obrotu i translacji
context.rotate(-3 * Math.PI / 4);
context.translate(-300, -370);
```

Kod z listingu 1.9 narysuje dwa obrócone obrazy statków kosmicznych widoczne na rysunku 1.7.



Rysunek 1.7. *Obracanie obrazów*

■ **Uwaga:** oprócz odwróconego obracania i translacji można też przywrócić stan obiektu `canvas`, najpierw wywołując metodę `save()` przed rozpoczęciem transformacji, a następnie metodę `restore()` na końcu transformacji.

Tym ostatnim przykładem zakończyliśmy omawianie wszystkich podstawowych zagadnień dotyczących elementu `canvas` potrzebnych do tworzenia gier. Nie omówiliśmy jednak wielu metod interfejsu API obiektu `canvas`, z którymi można się zapoznać pod adresem *https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API*.

Element audio

Korzystanie z elementu `audio` języka HTML5 jest nowym standardem umieszczania pliku dźwiękowego na stronie WWW. Zanim udostępniono ten element, większość stron odtwarzała pliki dźwiękowe za pomocą osadzonych dodatków (takich jak Flash).

Element `audio` można utworzyć w pliku HTML przy użyciu znacznika `<audio>`. Inny sposób polega na zastosowaniu kodu JavaScriptu i utworzeniu obiektu `Audio`. Listing 1.10 zawiera przykład.

Listing 1.10. *Znacznik <audio> języka HTML5*

```
<audio src="music.mp3" controls="controls">
    Twoja przeglądarka nie obsługuje elementu audio języka HTML5.
    ↪Prosimy o skorzystanie z nowszej przeglądarki.
</audio>
```

■ **Uwaga:** przeglądarki, które nie wspierają elementu `audio`, będą ignorować znacznik `<audio>` i renderować całą jego zawartość. Za pomocą tej funkcji możemy zapewnić alternatywną zawartość użytkownikom starszych przeglądarek lub komunikat, w którym poprosimy o użycie nowszej przeglądarki.

Atrybut `controls` (patrz listing 1.10) sprawia, że przeglądarka wyświetli prosty interfejs zależny od przeglądarki, umożliwiający odtwarzanie pliku wideo (taki jak przycisk odtwórz/zatrzymaj oraz kontrolki głośności).

Element `audio` ma kilka innych atrybutów, takich jak poniżej:

- `preload` — określa, czy należy wstępnie wczytać plik audio, czy nie;
- `autoplay` — określa, czy trzeba rozpocząć odtwarzanie dźwięku od razu po wczytaniu tego obiektu;
- `loop` — określa, czy po zakończeniu należy powtarzać odtwarzanie dźwięku.

Obecnie przeglądarki wspierają trzy popularne formaty plików: MP3 (MPEG Audio Layer 3), WAV (Waveform Audio) i OGG (Ogg Vorbis). Warto jednak zwrócić uwagę na to, że nie wszystkie przeglądarki wspierają wszystkie formaty dźwiękowe. Przykładowo: Firefox nie odtwarza plików MP3 bezpośrednio, ze względu na aspekty związane z patentami i licencjami (dlatego musi korzystać z wsparcia systemu operacyjnego), chociaż bezpośrednio odtwarza pliki OGG i WAV. Natomiast Safari wspiera MP3, ale nie wspiera plików OGG. W tabeli 1.1 zebrano formaty wspierane przez ostatnie wersje popularnych przeglądarek.

Tabela 1.1. *Formaty dźwiękowe wspierane przez obecnie dostępne przeglądarki*

Przeglądarka	MP3	WAV	OGG
Internet Explorer	Tak	Nie	Nie
Edge	Tak	Tak	Nie
Firefox	Z wykorzystaniem wsparcia systemu operacyjnego	Tak	Tak
Chrome	Tak	Tak	Tak
Safari	Tak	Tak	Nie
Opera	Tak	Tak	Tak

Aby zagwarantować odtwarzanie dźwięku w każdej przeglądarce, konieczne jest zapewnienie alternatywnych formatów naszego pliku. Element `audio` pozwala na podanie wielu elementów źródłowych w znaczniku `<audio>`, a przeglądarka automatycznie skorzysta z pierwszego rozpoznanego formatu (patrz listing 1.11).

Listing 1.11. *Znacznik <audio> z wieloma plikami źródłowymi*

```
<audio controls="controls">
    <source src="music.ogg" type="audio/ogg" />
    <source src="music.mp3" type="audio/mpeg" />
      Twoja przeglądarka nie wspiera elementu audio języka HTML5.
      ↪Prosimy o skorzystanie z nowszej przeglądarki.
</audio>
```

Dźwięk można też wczytać dynamicznie za pomocą obiektu `Audio` języka JavaScript. Obiekt `Audio` umożliwia wczytywanie, odtwarzanie i wstrzymywanie plików dźwiękowych, co przyda się nam w grach (patrz listing 1.12).

Listing 1.12. *Dynamiczne wczytywanie pliku dźwiękowego*

```
<script>
    // Tworzenie nowego obiektu Audio
    var sound = new Audio();

    // Wybieranie źródła dźwięku
    sound.src = "music.ogg";
    // Ten kod zadziała tylko w przeglądarkach wspierających format OGG

    // Odtwarzanie dźwięku
    // sound.play();
</script>
```

W przeciwieństwie do znacznika `<audio>` języka HTML, w którym można umieścić wiele formatów, podczas korzystania z języka JavaScript musimy znaleźć sposób na wykrycie formatów wspieranych przez przeglądarkę, aby móc wczytać odpowiedni format. Obiekt `Audio` oferuje metodę o nazwie `canPlayType()`, która zwraca wartości `""`, `"maybe"` lub `"probably"` określające wsparcie dla określonego kodeka. W ten sposób możemy z łatwością sprawdzić i wczytać potrzebny format pliku dźwiękowego, zgodnie z listingiem 1.13.

Listing 1.13. *Testowanie wsparcia dla formatów dźwiękowych*

```
<script>
    var audio = document.createElement("audio");
    var mp3Support, oggSupport;

    if (audio.canPlayType) {
        // Obecnie metoda canPlayType() zwraca: "", "maybe" lub "probably"
        mp3Support = "" !== audio.canPlayType("audio/mpeg");
        oggSupport = "" !== audio.canPlayType("audio/ogg; codecs=\"vorbis\"");
    } else {
        // Brak wsparcia dla znacznika audio
        mp3Support = false;
        oggSupport = false;
    }

    // Sprawdzamy wsparcie formatu ogg, następnie mp3, a w razie niepowodzenia ustawiamy wartość zmiennej
    // soundFileExtn na undefined
    var soundFileExtn = oggSupport ? ".ogg" : mp3Support ? ".mp3" : undefined;

    if (soundFileExtn) {
        var sound = new Audio();
        // Wczytujemy plik dźwiękowy w wykrytej wersji

        sound.src = "music" + soundFileExtn;
        sound.play();
    }
</script>
```

W listingu 1.13 wykorzystujemy metodę `canPlayType()`, aby ustawić właściwość `soundFileExtn`, za pomocą której możemy wczytać w przyszłości pliki dźwiękowe. Skorzystamy z tego pomysłu w dalszych rozdziałach, gdy będziemy dodawać obsługę dźwięków do naszych gier.

Obiekt `Audio` wyzwala kilka różnych zdarzeń, dzięki którym można stwierdzić, że dźwięk został wczytany i że jest gotowy do odtworzenia. Zdarzenie `loadedmetadata` jest wywoływane po wczytaniu początkowych metadanych pliku dźwiękowego do przeglądarki, natomiast zdarzenie `canplay` po pobraniu wystarczającego

fragmentu pliku dźwiękowego, aby można było rozpocząć odtwarzanie. Z kolei zdarzenie `canplaythrough` jest wyzwalane, gdy przeglądarka może odtworzyć cały plik dźwiękowy bez konieczności zatrzymywania odtwarzania i buforowania pliku. Za pomocą zdarzenia `canplaythrough` możemy śledzić, kiedy plik dźwiękowy został wczytany w wystarczającym stopniu, aby spełnić nasze potrzeby. Listing 1.14 zawiera przykład wykorzystania zdarzenia `canplaythrough` w celu odtworzenia dźwięku po jego wczytaniu.

Listing 1.14. *Czekanie na wczytanie pliku dźwiękowego*

```
<script>
    // Odtwarzamy dźwięk po odczekaniu na jego wczytanie
    if (soundFileExtn) {
        var sound = new Audio();

        sound.addEventListener("canplaythrough", function() {
            sound.play();
        });

        // Wczytujemy plik dźwiękowy z wykrytym rozszerzeniem
        sound.src = "music" + soundFileExtn;
    }
</script>
```

Dowiedzieliśmy się już, jak sprawdzać wspierane formaty plików dźwiękowych, jak dynamicznie wczytywać pliki dźwiękowe i wykrywać, kiedy został wczytany plik dźwiękowy. Te koncepcje możemy połączyć w projekt narzędzia do wstępnego wczytania dźwięku, które dynamicznie wczyta wszystkie zasoby dźwiękowe gry przed jej rozpoczęciem. W kilku kolejnych rozdziałach zapoznamy się bardziej szczegółowo z tą koncepcją podczas tworzenia narzędzia do wczytywania zasobów dla naszych gier.

Element image

Element `image` umożliwia wyświetlanie obrazów w pliku HTML. Najprościej można to wykonać za pomocą znacznika `<image>` i atrybutu `src`, co pokazano wcześniej, a także w poniższym listingu 1.15.

Listing 1.15. *Znacznik <image>*

```
<img src="spaceship.png" id="spaceship">
```

Obraz można też wczytać dynamicznie, za pomocą języka JavaScript, inicjalizując nowy obiekt `Image` i ustawiając jego właściwość `src`, zgodnie z listingiem 1.16.

Listing 1.16. *Dynamiczne wczytywanie obrazu*

```
var image = new Image();
image.src = "spaceship.png";
```

Aby uzyskać obraz do narysowania w obiekcie `canvas`, możemy skorzystać z dowolnej z tych metod.

Wczytywanie obrazów

Projektując gry, zwykle chcemy najpierw poczekać na całkowite wczytanie wszystkich obrazów, a dopiero potem inicjalizujemy pozostałe funkcje, aby uniknąć błędów wynikających z niedostatecznie wczytanych obrazów. Podczas wczytywania obrazów programiści zwykle wyświetlają pasek postępu lub wskaźnik stanu, który pokazuje procentowy stopień ukończenia tego procesu.

Obiekt `Image` oferuje zdarzenie `onload`, które jest wyzwalane od razu po zakończeniu wczytywania pliku obrazu przez przeglądarkę. Za pomocą tego zdarzenia możemy śledzić, kiedy obraz zostanie wczytany. Listing 1.17 zawiera przykładowe rozwiązanie.

Listing 1.17. *Oczekiwanie na wczytanie obrazu*

```
image.onload = function() {
    alert("Obraz został wczytany");
};
```

Za pomocą zdarzenia `onload` możemy utworzyć proste narzędzie do wczytywania obrazów, które będzie śledzić wczytane dotychczas obrazy (patrz listing 1.18).

Listing 1.18. *Proste narzędzie do wczytywania obrazów*

```
var imageLoader = {
    loaded: true,
    loadedImages: 0,
    totalImages: 0,
    load: function(url) {
        this.totalImages++;
        this.loaded = false;
        var image = new Image();
        image.src = url;
        image.onload = function() {
            imageLoader.loadedImages++;
            if (imageLoader.loadedImages === imageLoader.totalImages) {
                imageLoader.loaded = true;
            }
            image.onload = undefined;
        }
        return image;
    }
}
```

W powyższym kodzie utworzyliśmy obiekt `imageLoader` zawierający metodę `load()`. Ta metoda przyjmuje adres URL obrazu i w każdym wywołaniu inkrementuje licznik `totalImages`. Następnie dynamicznie tworzy obiekt `Image` i ustawia jego właściwość `src`. Na końcu, w metodzie obiektu obsługującej zdarzenie `onload`, inkrementuje licznik `loadedImages`, a gdy ten będzie miał taką samą wartość jak `totalImages`, ustawia wartość zmiennej `loaded` na `true`.

Powyższe narzędzie do wczytywania obrazu można wykorzystać do wczytania wielu obrazów (np. w pętli). Za pomocą zmiennej `imageLoader.loaded` możemy sprawdzić, czy wszystkie obrazy zostały wczytane. Natomiast na podstawie wartości `loadedImages/totalImages` jesteśmy w stanie wyświetlić pasek postępu.

Nie należy się na razie przejmować użyciem tego narzędzia. Jest to jedynie fragment kodu, który ułatwia zrozumienie podstawowej zasady działania narzędzia do wczytywania obrazów. W kolejnych rozdziałach utworzymy bardziej kompletną wersję narzędzia do wczytywania zasobów, z którego będziemy mogli korzystać w naszych grach.

Arkusze sprite'ów

Innym problemem, zwłaszcza w grze zawierającej dużą liczbę obrazów, jest sposób optymalizacji sposobu ich wczytywania przez serwer. Gry mogą wymagać wczytania dziesiątek albo i setek obrazów. Nawet prosta gra strategiczna w czasie rzeczywistym (RTS) wymaga obrazów różnych jednostek, budynków, map, tła oraz efektów. W przypadku jednostek i budynków możemy potrzebować wielu wersji obrazów, które będą reprezentować różne kierunki i stany, a w przypadku animacji wymagany będzie obraz dla każdej klatki.

W jednym z moich wcześniejszych projektów gry RTS używałem pojedynczych obrazów dla poszczególnych ramek każdej animacji oraz stanu każdej jednostki i budynku. Liczba potrzebnych obrazów przekroczyła 1000. Ponieważ większość przeglądarek wykonuje tylko kilka żądań jednocześnie, pobieranie tych wszystkich obrazów zajmowało bardzo dużo czasu, doprowadzając zarazem do przeciążenia serwera. Chociaż nie stanowiło to problemu podczas testowania kodu lokalnie, okazało się jednak rodzić trudności po umieszczeniu kodu na serwerze. Gracze musieli czekać od 5 do 10 minut (czasem dłużej) na wczytanie gry, zanim mogli zacząć zabawę. Wszystkie jednoczesne żądania przyczyniły się też do znacznego obciążenia mojego serwera WWW.

Na szczęście istnieje prosty sposób na rozwiązanie problemu ze zbyt dużą liczbą obrazów i żądań HTTP. Są to arkusze sprite'ów. Te arkusze w jednym dużym pliku graficznym zawierają wszystkie sprite'y (obrazy), potrzebne w grze. Podczas wyświetlania obrazów obliczamy przesunięcie sprite'a, który chcemy wyświetlić, i za pomocą metody `drawImage()` rysujemy tylko fragment obrazu. Obraz *spaceship.png*, z którego korzystamy w tym rozdziale, jest przykładem arkusza sprite'ów, ponieważ w jednym pliku znajduje się wiele sprite'ów statku kosmicznego.

Analizując kod z listingów 1.19 i 1.20, możemy poznać przykłady rysowania obrazu wczytanego pojedynczo i obrazu wczytanego z arkusza sprite'ów.

Listing 1.19. *Rysowanie pojedynczego wczytanego obrazu*

```
// Najpierw: (wczytujemy poszczególne obrazy i zapisujemy je w dużej tablicy)

// Trzy argumenty: element oraz współrzędne docelowe (x, y)
var image = imageArray[imageNumber];
context.drawImage(image, x, y);
```

Listing 1.20. *Rysowanie obrazu wczytanego z arkusza sprite'ów*

```
// Najpierw: (wczytujemy jeden obraz arkusza sprite'ów)
// Dziewięć argumentów: element, współrzędne źródłowe (x, y),
// szerokość i wysokość źródłowa (potrzebna do kadrowania),
// współrzędne docelowe (x, y) oraz
// docelowa szerokość i wysokość (zmiana rozmiaru)

context.drawImage (this.spriteImage, this.imageWidth*(imageNumber), 0, this.imageWidth,
↪this.imageHeight, x, y, this.imageWidth, this.imageHeight);
```

W pierwszym przykładzie zapisujemy każdy sprite w osobnym obiekcie `Image` umieszczonym w tablicy, a następnie wyświetlamy go, odwołując się do obiektu `Image`. Ta metoda wymaga utworzenia tylu obiektów `Image`, ile sprite'ów potrzebujemy, a także tyle samo żądań HTTP wysyłanych do serwera w celu uzyskania każdego obrazu.

W drugim przykładzie wczytujemy jeden ogromny arkusz sprite'ów, na którym znajdują się obok siebie wszystkie sprite'y. Rysowanie każdego sprite'a wymaga obliczenia przesunięcia względem osi x i y tego sprite'a w obrazie, a dalej naszkicowania odpowiedniego fragmentu obrazu. Ta metoda wymaga tylko jednego żądania HTTP i tylko jednego obiektu `Image` dla całego arkusza sprite'ów. Ponadto musimy wykonać nieco więcej obliczeń, aby wyznaczyć położenie sprite'a na obrazie. Jeśli chodzi o wykorzystanie zasobów sieciowych, jest to znacznie wydajniejsza metoda.

Poniżej opisano kilka zalet korzystania z arkusza sprite'ów, dzięki którym użycie opisanej metody w dowolnej złożonej grze okaże się bardzo łatwe:

- *Mniej żądań HTTP* — jednostka, która wymaga 80 obrazów (a zatem 80 żądań) zostanie pobrana w jednym żądaniu HTTP.
- *Lepsza kompresja* — przechowywanie obrazów w jednym pliku oznacza, że informacje z nagłówka nie powtarzają się w każdym pliku, a rozmiar jednego złożonego pliku jest znacznie mniejszy niż łączny rozmiar wszystkich poszczególnych plików.

- *Szybszy czas ładowania* — dzięki znacznie mniejszej liczbie żądań HTTP oraz mniejszemu rozmiarowi plików użycie przepustowości i czas ładowania w grze również spadają, co oznacza, że nie będzie trzeba tak długo czekać na wczytanie gry.

Animacja: czasomierz i pętle gry

Zanim przejdziemy do tworzenia gier, musimy się jeszcze zapoznać z zagadnieniem animacji. Animacja polega na rysowaniu obiektu, wymazywaniu go i ponownym rysowaniu w nowym położeniu, a wszystko to wystarczająco szybko, aby oko ludzkie postrzegało te zmiany jak płynny ruch.

Najpopularniejszym sposobem obsługi animacji jest utworzenie funkcji rysującej, która jest wywoływana wiele razy na sekundę. W tej funkcji iterujemy wszystkie encje gry i rysujemy je po kolei.

W prostszych grach zwykle do tworzenia animacji oraz przesuwania encji i ich rysowania służy ta sama funkcja rysująca. Jednakże niektóre gry mają osobne funkcje służące do animacji, które aktualizują ruch encji w grze, natomiast funkcja rysująca obsługuje tylko rysowanie encji na ekranie. Ponieważ funkcja animacji jest niezależna od funkcji rysującej, można ją wywoływać rzadziej niż funkcję rysującą. Listing 1.21 zawiera szkieletowy kod z typowymi procedurami animacji i rysowania.

Listing 1.21. *Typowa pętla animacji i rysowania*

```
function animationLoop(){
    // Iterowanie wszystkich elementów występujących w grze
    // i ich przesuwanie
}

function drawingLoop(){
    // 1. Czyszczenie elementu canvas
    // 2. Iterowanie wszystkich elementów
    // 3. Rysowanie każdego z nich
}
```

Zakładając, że w swojej grze korzystamy z metody `drawingLoop()`, musimy opracować sposób powtarzalnego wywoływania tej metody w regularnych odstępach czasu. Najprościej można to wykonać za pomocą dwóch metod czasomierza `setInterval()` i `setTimeout()`. Metoda `setInterval(functionName, timeInterval)` informuje przeglądarkę, że należy wywoływać podaną funkcję w stałych odstępach czasowych, aż zostanie wywołana funkcja `clearInterval()`. Jeśli chcemy zatrzymać animowanie (gdy gra zostanie przerwana lub się zakończy), korzystamy z metody `clearInterval()`. Listing 1.22 przedstawia przykład korzystania z tych funkcji.

Listing 1.22. *Wywoływanie pętli rysowania za pomocą funkcji setInterval()*

```
// Wywołujemy metodę drawingLoop() co 20 millisekund
var gameLoop = setInterval(drawingLoop, 20);
// Zatrzymujemy wywoływanie metody drawingLoop() i czyścimy zmienną gameLoop
clearInterval(gameLoop);
```

Metoda `setTimeout(functionName, timeInterval)` informuje przeglądarkę, że trzeba wywołać podaną funkcję jeden raz, po upływie podanego interwału czasowego. Przykład znajduje się w listingu 1.23.

Listing 1.23. *Wywoływanie pętli rysowania za pomocą funkcji setTimeout()*

```
function drawingLoop(){
    // 1. Wywołujemy metodę drawingLoop() jeden raz, po 20 milisekundach
    var gameLoop = setTimeout(drawingLoop,20);

    // 2. Czyścimy element canvas
```

```
    // 3. Iterujemy wszystkie elementy

    // 4. Rysujemy je
}
```

W przeciwieństwie do metody `setInterval()` podczas korzystania z metody `setTimeout()` musimy za każdym razem wykonać nowe wywołanie, ponieważ ta metoda wywołuje funkcję `drawingLoop()` tylko jeden raz. Jeśli chcemy zatrzymać animację (po wstrzymaniu lub zakończeniu gry), możemy skorzystać z metody `clearTimeout()`:

```
// Przestajemy wywoływać metodę drawingLoop() i czyścimy zmienną gameLoop
clearTimeout(gameLoop);
```

Nie należy się zbyt przejmować, jeśli na tym etapie niektóre z omówionych zagadnień są niezbyt zrozumiałe lub abstrakcyjne. Ten rozdział ma na celu zapewnienie jedynie szybkiego wprowadzenia. Wystarczy jedynie zrozumieć ogólne zasady działania opisanych tu mechanizmów. Gdy w kolejnych rozdziałach zaczniemy tworzyć gry, poznamy szczegółowe przykłady użycia tych funkcji. Wtedy wszystko stanie się bardziej zrozumiałe.

requestAnimationFrame

Chociaż metody `setInterval()` lub `setTimeout()` dobrze sprawdzają się przy tworzeniu animacji, producenci przeglądarek opracowali nowy interfejs API przeznaczony do obsługiwania animacji. Dzięki użyciu tego interfejsu zamiast funkcji `setInterval()` przeglądarka może:

- Zoptymalizować kod animacji w jednym cyklu zmiany układu elementów i ich ponownego narysowania, co prowadzi do uzyskania płynniejszej animacji.
- Wstrzymać animację, gdy zakładka z grą nie jest widoczna, co prowadzi do mniejszego użycia procesora i karty graficznej.
- Automatycznie ograniczyć częstotliwość odświeżania klatek na komputerach, które nie wspierają wyższych częstotliwości, lub zwiększyć częstotliwość na komputerach, które mogą je przetworzyć.

W czasie pisania pierwszej wersji tej książki producenci przeglądarek korzystali ze wspomnianego interfejsu za pomocą własnych metod (np. firma Microsoft stosowała metodę `msrequestAnimationFrame()`, a Mozilla metodę `mozRequestAnimationFrame()`). Jednak od pewnego czasu wszystkie przeglądarki dokonały standaryzacji implementacji tego interfejsu API i obecnie z metod `requestAnimationFrame()` i `cancelAnimationFrame()` można korzystać we wszystkich przeglądarkach wspierających HTML5.

Uwaga: obecnie nie ma gwarancji określonej częstotliwości odświeżania klatek (przeglądarka decyduje o szybkości wywoływania pętli rysowania), dlatego musimy zapewnić, że obiekty animowane poruszają się z taką samą prędkością na ekranie, niezależnie od rzeczywistej częstotliwości odświeżania klatek. W tym celu animujemy obiekty w osobnych pętlach `setTimeout()` lub `setInterval()` albo obliczamy czas od ostatniego cyklu rysowania i za jego pomocą dokonujemy interpolacji położenia animowanego obiektu.

Metodę `requestAnimationFrame()` można wywołać w metodzie `drawingLoop()`, podobnie jak w metodzie `setTimeout()`, co zaprezentowano w listingu 1.24.

Listing 1.24. *Wywoływanie pętli rysowania za pomocą metody requestAnimationFrame()*

```
function drawingLoop(nowTime){
    // 1. Wywołujemy metodę drawingLoop(), gdy tylko przeglądarka jest gotowa na ponowne rysowanie
    var gameLoop = requestAnimationFrame(drawingLoop);

    // 2. Czyścimy obiekt canvas
```

```
    // 3. Iterujemy wszystkie elementy

    // 4. Opcjonalnie używamy bieżącej wartości nowTime oraz poprzedniej wartości nowTime, aby przeprowadzić
    // interpolację klatek

    // 5. Rysujemy elementy
}
```

Jeśli musimy zatrzymać animację (podczas wstrzymania lub zakończenia gry), możemy skorzystać z metody `cancelAnimationFrame()`:

```
// Przestajemy wywoływać metodę drawingLoop() i czyścimy zmienną gameLoop
cancelAnimationFrame(gameLoop);
```

W tym podrozdziale omówiliśmy główne sposoby dodawania animacji do gier. W kolejnych rozdziałach będziemy się zajmować rzeczywistą implementacją tych pętli animacji.

Podsumowanie

W tym rozdziale zapoznaliśmy się z podstawowymi elementami HTML5 potrzebnymi do tworzenia gier. Nauczyliśmy się używać elementu `canvas` w celu rysowania kształtów, pisania tekstu i manipulowania obrazami. Sprawdziliśmy, jak używać elementu audio, aby wczytać i odtworzyć dźwięki w różnych przeglądarkach. Omówiliśmy też pokrótce podstawy tworzenia animacji, wczytując wcześniej obiekty i korzystając z arkuszy sprite'ów.

Omówione tu zagadnienia są jedynie punktem wyjścia i zdecydowanie nie zostały omówione wyczerpująco. Ten rozdział miał na celu wprowadzenie do tych zagadnień lub przypomnienie wiadomości o HTML5. Można go również potraktować jako przydatny przewodnik, w którym z łatwością da się znaleźć omówienie składni oraz przykładowy kod. Jak wcześniej wspomniano, podczas tworzenia naszych gier w kolejnych rozdziałach przyjrzymy się tym zagadnieniom bardziej szczegółowo.

Jeśli ktoś nadal nie czuje się zbyt pewnie i chciałby uzyskać bardziej szczegółowe informacje o podstawach języków JavaScript i HTML5, zalecam przeczytanie książek wprowadzających w tajniki tych języków, np. *JavaScript for Absolute Beginners* Terry'ego McNavage'a oraz *The Essential Guide to HTML5* Jeanine Meyer.

Po zapoznaniu się z podstawowymi zagadnieniami możemy zacząć pisanie naszej pierwszej gry.

Skorowidz

E

F

G

H

I

K

L

Ł

M

N

O

P

R

U

V

W

Z


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

HTML5 — idealny język dla profesjonalnego twórcy gier!

HTML5 jest dziś kojarzony ze znakomitym narzędziem do tworzenia zaawansowanych, interaktywnych i dynamicznych aplikacji internetowych. Udostępnia programistom szereg nowoczesnych technologii, które są konsekwentnie rozwijane w ramach wielu specyfikacji nadzorowanych przez W3C i grupę WHATWG. Szczególnie atrakcyjne jest wykorzystanie HTML5 do tworzenia gier. Mogą to być gry najróżniejszych rodzajów: od prostych układanek po skomplikowane gry fabularne dla wielu graczy. Aby jednak Twoje dzieło stało się prawdziwym hitem, musisz się dowiedzieć, jak swoją pracę wykonać na wysokim, profesjonalnym poziomie.

Ta książka jest drugim, uaktualnionym i uzupełnionym, wydaniem świetnego podręcznika dla programistów gier. Dzięki niej zaczniesz mistrzowsko stosować zaawansowane techniki programistyczne w języku HTML5. Nauka będzie polegać na wykonaniu dwóch projektów: strategicznej gry planszowej oraz gry czasu rzeczywistego typu RST. Dzięki przejrzystym wskazówkom i dokładnym instrukcjom sprawnie ukończysz kolejne elementy aplikacji. W ten sposób zapoznasz się z najważniejszymi narzędziami służącymi do tworzenia gier, a także dowiesz się, w jaki sposób praktycznie wykorzystać najbardziej zaawansowane możliwości HTML. Dzięki książce nauczysz się również pisać gry na urządzenia mobilne. Szybko zaczniesz tworzyć złożone i dopracowane gry i staniesz się profesjonalnym programistą gier HTML5.

W tej książce między innymi:

- wykorzystanie silnika fizyki Box2D i uzyskiwanie realistycznych efektów
- projektowanie złożonych światów i interaktywnych postaci
- efekty paralaksy i efekty dźwiękowe
- algorytmy odnajdowania ścieżek oraz nawigacji
- wykorzystanie drzew decyzyjnych, maszyn stanów oraz zdarzeń skryptowych

Aditya Ravi Shankar — zaczął programować w 1993 roku. Przez prawie dziesięć lat zajmował się systemami sprzedaży i analizy dla banków inwestycyjnych oraz dla dużych firm z listy Fortune 100. Później zaczął tworzyć własne projekty oraz eksperymentować chyba ze wszystkimi możliwymi językami programowania i technologiami, specjalizując się w pisaniu najróżniejszych gier. W wolnych chwilach pomaga rozmaitym firmom we wdrażaniu nowego oprogramowania.

ISBN 978-83-283-4325-2