

WPROWADZENIE DO

HTML5

Nauka HTML5 i JavaScriptu na przykładzie gier

JEANINE MEYER

Helion



Tytuł oryginału: The Essential Guide to HTML5: Using Games to learn HTML5 and JavaScript

Tłumaczenie: Marek Pętlicki

ISBN: 978-83-246-4107-9

Original edition copyright 2010 by Jeanine Meyer.
All rights reserved.

Polish edition copyright 2012 by HELION SA.
All rights reserved.

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

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

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

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
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/whtmjs.zip>

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

Printed in Poland.

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

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

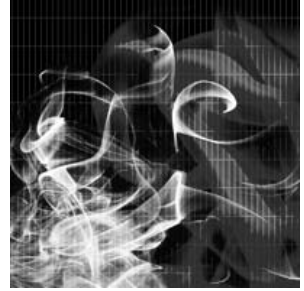
Spis treści

O autorce	9
Redaktor techniczny	10
Wstęp	11
Rozdział 1. Podstawy	15
Wprowadzenie	15
Wymagania	16
Funkcje HTML5, CSS i JavaScriptu	18
Podstawowe struktury i znaczniki HTML	18
Programowanie w JavaScriptcie	24
Zbuduj aplikację i weź ją dla siebie	26
Testowanie aplikacji i wrzucanie jej na serwer	32
Podsumowanie	32
Rozdział 2. Gra w kości	35
Wprowadzenie	35
Wymagania	38
Funkcje HTML5, CSS i JavaScriptu	38
Przetwarzanie liczb pseudolosowych i wyrażenia matematyczne	38
Zmienne i instrukcje przypisania	40
Funkcje użytkownika	41
Kontrola logiki kodu: instrukcje if i switch	42
Rysowanie na elemencie canvas	45
Zbuduj własną aplikację	53
Rzut pojedynczą kością	55
Rzut dwiema kośćmi	60
Kompletna gra w kości	65
Testowanie aplikacji i wrzucenie jej na serwer	72
Podsumowanie	73

Rozdział 3. Odbijająca się piłka	75
Wprowadzenie	75
Wymagania	78
Funkcje HTML5, CSS i JavaScriptu	78
Rysowanie piłki, ilustracji i gradientów	79
Zbuduj aplikację i weź ją dla siebie	90
Testowanie aplikacji i wrzucenie jej na serwer	100
Podsumowanie	100
Rozdział 4. Armata i proca	103
Wprowadzenie	103
Wymagania	106
Funkcje HTML5, CSS i JavaScriptu	107
Tablice i obiekty definiowane przez programistę	107
Obroty i przesunięcia w operacjach rysowania	109
Rysowanie odcinków	113
Zdarzenia myszy: napinanie procy	114
Modyfikowanie listy wyświetlanych elementów z użyciem metody splice()	116
Odległość między punktami	117
Zbuduj aplikację i weź ją dla siebie	117
Strzał z armaty: ustawianie kąta i prędkości	122
Proca: definiowanie parametrów lotu pocisku za pomocą myszy	128
Testowanie aplikacji i wrzucenie jej na serwer	136
Podsumowanie	137
Rozdział 5. Pamięć	139
Wprowadzenie	139
Podstawowe wymagania	143
Funkcje HTML5, CSS i JavaScriptu	144
Reprezentowanie kart	144
Wykorzystanie obiektu Date do obliczania czasu	146
Obsługa pauzy	146
Rysowanie tekstu	147
Rysowanie wielokątów	149
Mieszanie kart	150
Kliknięcia w karty	151
Zapobieganie oszustwom	152
Zbuduj własną aplikację	153
Testowanie aplikacji i wrzucenie jej na serwer	166
Podsumowanie	166

Rozdział 6. Quiz	167
Wprowadzenie	167
Podstawowe wymagania	172
Funkcje HTML5, CSS i JavaScriptu	172
Zapisywanie informacji w tablicach i ich odczytywanie	172
Tworzenie elementów HTML w czasie działania programu	175
Zmiana stylu CSS elementów za pomocą kodu JavaScript	177
Używanie formularzy i ich pól do komunikacji z graczem	179
Wyświetlanie wideo	180
Zbuduj własną aplikację	182
Testowanie aplikacji i wrzucenie jej na serwer	193
Podsumowanie	194
Rozdział 7. Labirynt	195
Wprowadzenie	195
Wymagania podstawowe	201
Funkcje HTML5, CSS i JavaScriptu	201
Reprezentacja ścian i pionka	201
Wykorzystanie zdarzeń myszy do budowania ścian	202
Obsługa klawiszy strzałek	203
Detekcja kolizji: pionek i ściany	204
Wykorzystanie lokalnego magazynu danych	207
Kodowanie danych do zapisu w magazynie lokalnym	212
Przyciski typu radio	214
Zbuduj własną aplikację	214
Druga wersja aplikacji „Labirynt”	224
Testowanie aplikacji i wrzucenie jej na serwer	231
Podsumowanie	231
Rozdział 8. Kamień, papier, nożyce	233
Wprowadzenie	233
Podstawowe wymagania	236
Funkcje HTML5, CSS i JavaScriptu	237
Tworzenie graficznych przycisków do interakcji z graczem	237
Generowanie ruchu komputera	241
Zaczynamy	249
Zbuduj własną aplikację	250
Testowanie aplikacji i wrzucenie jej na serwer	257
Podsumowanie	257

Rozdział 9. Wisielec	259
Wprowadzenie	259
Wymagania podstawowe	265
Funkcje HTML5, CSS i JavaScriptu	265
Przechowywanie listy wyrazów w tablicy zdefiniowanej w zewnętrznym skrypcie	266
Generowanie, pozycjonowanie elementów HTML, formatowanie elementów w postać przycisków i blokowanie przycisków	266
Tworzenie rysunku na elemencie canvas	269
Śledzenie stanu gry i rozstrzygnięcie wygranej lub przegranej	271
Sprawdzenie, czy gracz odgadł, i ustawienie textContent	272
Zbuduj własną aplikację	273
Testowanie aplikacji i wrzucenie jej na serwer	282
Podsumowanie	282
Rozdział 10. Blackjack	283
Wprowadzenie	283
Wymagania podstawowe	288
Funkcje HTML5, CSS i JavaScriptu	289
Zbuduj własną aplikację	297
Testowanie aplikacji i wrzucenie jej na serwer	306
Podsumowanie	307
Skorowidz	308



Rozdział 5

Pamięć

W tym rozdziale omówimy następujące zagadnienia:

- rysowanie wielokątów;
- wypisywanie tekstów na elemencie canvas;
- techniki programistyczne reprezentowania informacji;
- programowanie przerwy;
- obliczanie minionego czasu;
- metoda mieszania kart.

Wprowadzenie

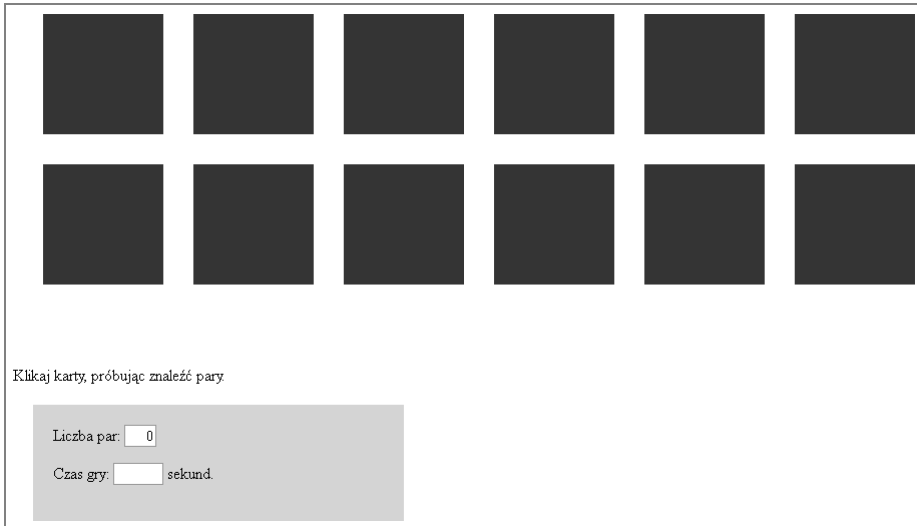
Ten rozdział demonstruje dwie wersje gry karcianej znanej jako gra w pamięć (ang. *memory game*). Karty są układane na stole koszulką do góry, a gracz odkrywa dwie naraz (klikając je myszą), starając się dopasować pary. Program usuwa dopasowania ze stołu, ale jeśli karty nie są dopasowane, odwraca je znów koszulkami do góry. Gdy gracz znajdzie wszystkie dopasowania, gra prezentuje wynik w postaci czasu trwania gry.

Pierwsza wersja gry będzie na kartach rysować wielokąty, druga wykorzystuje zdjęcia. Między tymi wersjami znajdziesz też więcej różnic, które mają na celu zobrazowanie funkcji HTML5, ale zachęcam do wyszukiwania nie tylko różnic, ale też jak największej liczby podobieństw tych aplikacji.

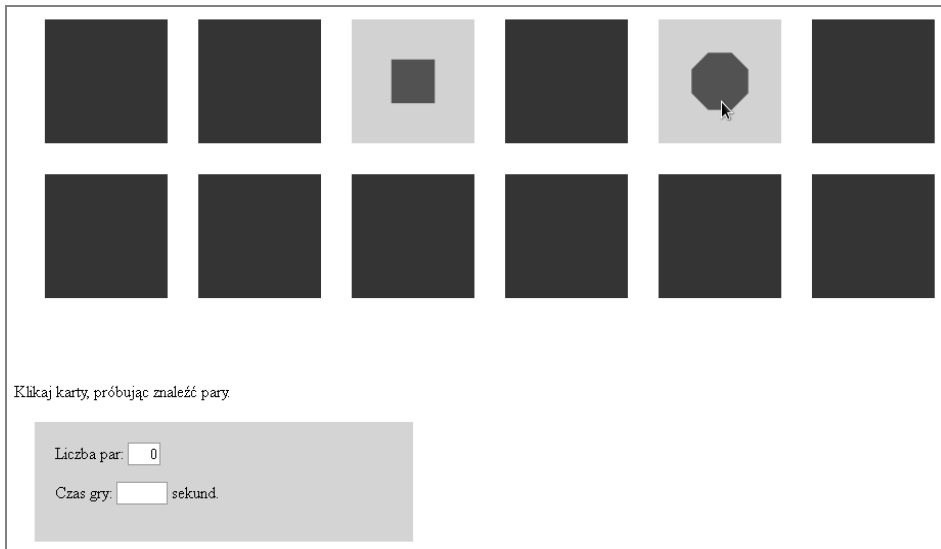
Rysunek 5.1 prezentuje początkowy ekran pierwszej wersji aplikacji. Gdy gracz ukończy grę, formularz prezentujący liczbę par wyświetli dodatkowo czas gry.

Rysunek 5.2 prezentuje efekt kliknięcia dwóch kart. Wielokąty na kartach nie pasują, więc po chwili przerwy program „odwraca” karty, wyświetlając ich koszulki.

Gdy dwie karty pasują do siebie, aplikacja usuwa je ze stołu i modyfikuje w formularzu liczbę dopasowań (rysunek 5.3).



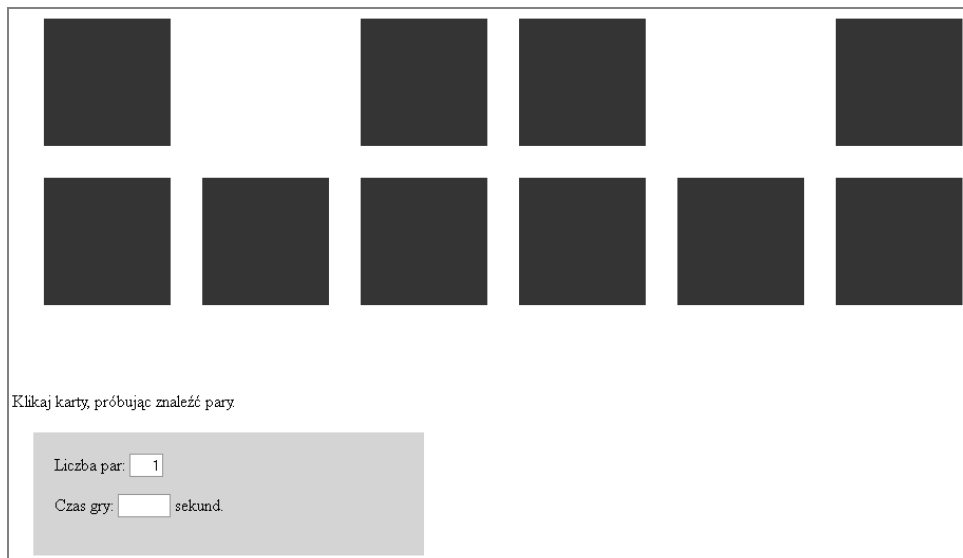
Rysunek 5.1. Ekran początkowy gry „Pamięć”



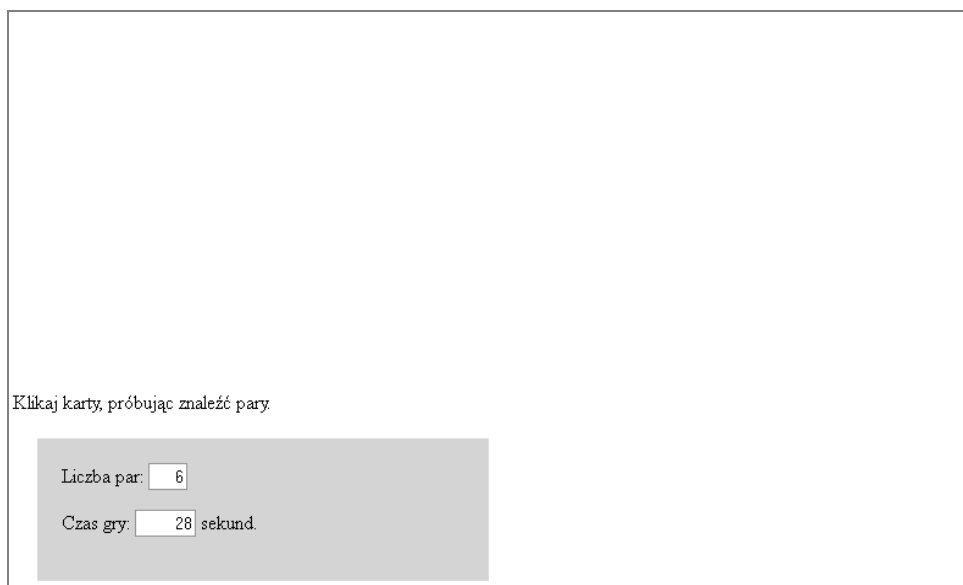
Rysunek 5.2. Dwie karty odkryte, nie ma pary

Rysunek 5.4 przedstawia wynik końcowy: w tym przypadku 6 dopasowań w czasie 28 sekund.

W drugiej wersji gry fronty kart zamiast wielokątów wykorzystują zdjęcia osób. Mimo że wiele gier tego typu dopasowania ma zdefiniowane jako identyczne obrazy, to w tym przypadku traktujemy tę sytuację jako „2 kier pasuje do 2 karo” (czerwone kolory) w zwykłej talii kart. Innymi słowy: dopasowanie jest zdefiniowane jako dwa różne zdjęcia tej samej osoby. Takie założenie gry wymaga zdefiniowania specjalnej struktury danych, która określi dopasowania. Ta wersja gry dodatkowo demonstruje technikę wypisywania tekstu na elemencie canvas, co przedstawia rysunek 5.5.



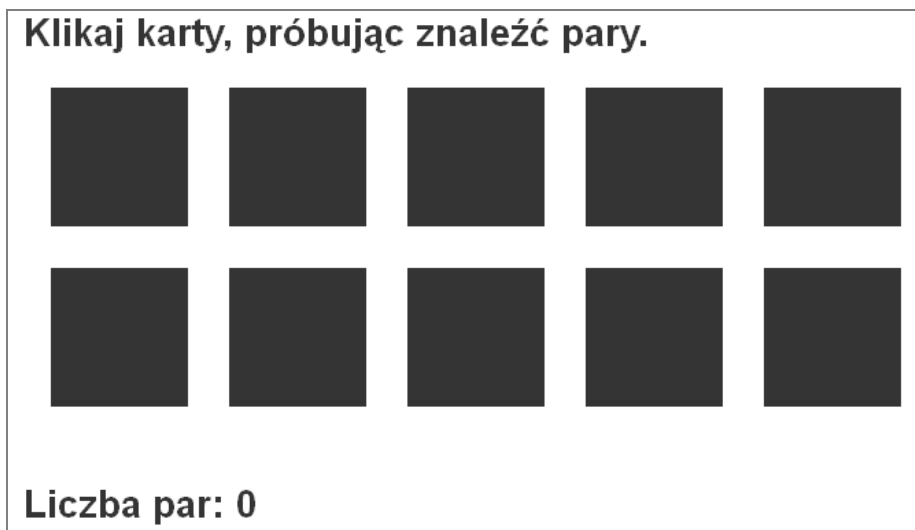
Rysunek 5.3. Aplikacja usunęła dwie dopasowane karty



Rysunek 5.4. Pierwsza wersja gry: widok ekranu po ukończeniu

Przykład sytuacji po kliknięciu dwóch kart prezentuje rysunek 5.6.

Ponieważ odkryte karty zawierają zdjęcia dwóch różnych osób, po krótkiej przerwie, pozwalającej graczowi przyrzeć się obrazom, gra odwraca karty i pozwala graczowi spróbować ponownie. Rysunek 5.7 przedstawia sytuację dopasowania: dwa różne zdjęcia tej samej osoby.



Rysunek 5.5. Druga wersja gry „Pamięć”, ekran początkowy



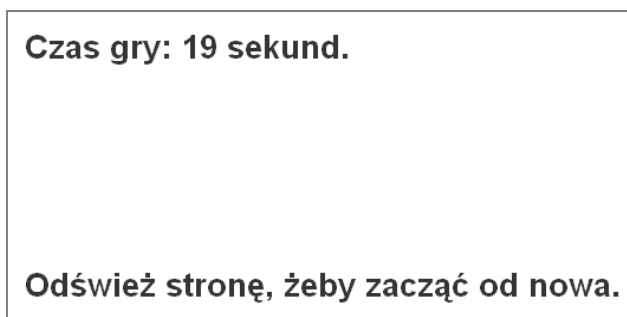
Rysunek 5.6. Dwie odkryte karty, brak dopasowania

Dopasowane zdjęcia są usuwane ze stołu. Gdy zostaną usunięte wszystkie karty, na ekranie zostaje wyświetlony całkowity czas gry wraz z instrukcją, w jaki sposób rozpocząć grę od nowa, co przedstawia rysunek 5.8.

W tę grę można grać, wykorzystując zdjęcia dostępne wraz z kodem HTML na serwerze FTP. Dużo więcej frajdy sprawia jednak gra ze zdjęciami znanych sobie osób. Można zacząć od zestawu dwóch-trzech par zdjęć i stopniowo skompletować zdjęcia całej rodziny, klasy albo klubu. W pierwszej wersji gry zamiast wielokątów można zaimplementować własne, bardziej efektowne kreacje.



Rysunek 5.7. Udana dopasowanie: dwa różne zdjęcia tej samej osoby



Rysunek 5.8. Ostatni ekran gry (druga wersja). Wszystkie zdjęcia zostały dopasowane, więc na stole nie ma już ani jednej karty

Podstawowe wymagania

Komputerowa wersja gry w karty wymaga reprezentowania koszulek (tylnej strony) kart, które są identyczne, oraz ich frontów. W naszych przykładach w tej drugiej roli zastosujemy wielokąty i zdjęcia. Aplikacja musi umieć stwierdzić, czy odkryte karty pasują do siebie i gdzie na stole znajdują się poszczególne karty. Dodatkowo gracz potrzebuje informacji zwrotnych dotyczących tego, co się dzieje w grze. W rzeczywistej grze gracze odwracają po dwie karty i sprawdzają, czy znaleźli dopasowanie, co zajmuje chwilę. Jeśli dopasowania nie ma, karty są ponownie odwracane koszulką do góry.

Program komputerowy musi wyświetlać fronty wybranych kart, a po odkryciu drugiej karty musi na chwilę zatrzymać działanie, aby gracz zobaczył, co zostało odkryte. Ta pauza jest przykładem działania podejmowanego przez komputer w celu zasymulowania sytuacji świata rzeczywistego, czyli

w tym przypadku prawdziwej gry w karty. Aplikacja powinna również wyświetlić aktualną liczbę znalezionych par, a po zakończeniu gry ilość czasu, jaki upłynął od jej rozpoczęcia. Wersje aplikacji wykorzystujące wielokąt i zdjęcia różnią się nieco w zakresie sposobu prezentowania tych informacji graczowi.

Oto podsumowanie funkcji, jakie muszą realizować te dwie gry:

- rysowanie koszulek kart;
- losowanie kolejności kart na stole w celu uniknięcia powtarzania się tego samego układu;
- wykrywanie kliknięcia w kartę i rozróżnianie między pierwszym a drugim kliknięciem;
- po rozpoznaniu kliknięcia wyświetlenie frontu właściwej karty przez narysowanie wielokąta (wersja pierwsza) lub wyświetlenie zdjęcia (wersja druga);
- usunięcie dopasowanych par;
- podejmowanie odpowiednich reakcji nawet w przypadku, gdy gracz wykonuje pozornie bezsensowne działania, jak dwukrotne kliknięcie tej samej karty lub pustego miejsca na stole, na którym wcześniej leżała karta.

Funkcje HTML5, CSS i JavaScriptu

Omówmy funkcje HTML5 i JavaScriptu specyficzne dla gier implementowanych w tym rozdziale. Ponownie naszą implementację oprzemy na wiedzy zdobytej do tej pory, dotyczącej ogólnej struktury dokumentów HTML, umiejętności rysowania na elemencie canvas czworokątów, obrazów i ścieżek złożonych z odcinków, wykorzystania formularzy i tablic.

Nowościami w tym rozdziale będą: wykorzystanie zdarzenia zaplanowanego w czasie, użycie obiektu Date do obliczania czasu, jaki upłynął, wypisywanie tekstu na elemencie canvas oraz kilka użytecznych technik programowania, które przydadzą się też w innych aplikacjach.

Podobnie jak w poprzednich rozdziałach, ten podrozdział omawia funkcje HTML oraz techniki programowania w ujęciu ogólnym. Kod przykładowych aplikacji znajduje się w podrozdziale „Zbuduj własną aplikację”. Jeśli chcesz, możesz przeskoczyć do tego podrozdziału i zobaczyć kod, ale sugeruję, żebyś wrócił w to miejsce, żeby zapoznać się z objaśnieniami technik użytych w kodzie.

Reprezentowanie kart

Każdy, kto trzymał w ręce prawdziwą kartę, wie jak ona wygląda. Z jednej strony znajduje się rysunek informujący o znaczeniu karty, a z drugiej wzór ozdobny (nazywany koszulką lub rewersem), który jest identyczny na wszystkich kartach z tej samej talii. Gdy rozłożymy karty na stole, wiemy doskonale, gdzie się znajdują, wiemy też, które karty są odwrócone frontem, a które koszulką do góry. Komputer musi te informacje zapisać w jakiejś strukturze (*zakodować*). Kodowanie informacji jest jednym z kluczowych elementów większości programów komputerowych, nie tylko gier.

W tym rozdziale (i w dalszej części książki) opiszę sposoby realizacji tego zadania. Należy pamiętać, że nie istnieje jedno, idealne podejście do tego problemu. Różne strategie budowania aplikacji jednak często bazują na podobnych założeniach.

Nasze podejście do zakodowania informacji o kartach opiera się na obiektach definiowanych przez programistę. Tworzenie takich obiektów w JavaScriptcie wiąże się z pisaniem funkcji konstruktorów; w naszym przypadku będzie to funkcja `Card()`. Zaleta wykorzystania obiektów polega na tym, że w języku JavaScript możemy w prosty sposób wykorzystać notację z kropką, uzyskując dostęp do różnych elementów obiektu (informacji). W podobny sposób wykorzystywaliśmy obiekty w rozdziale 4., przy okazji armaty i procy.

Obiekt `Card` będzie wyposażony we właściwości przechowujące informacje o położeniu karty na stole (`sx` i `sy`) i jej wymiarach (`swidth` i `sheight`), funkcję rysującą koszulkę karty na elemencie `canvas` oraz informację o tym, w jaki sposób rysować front karty.

W przypadku pierwszej wersji gry, w której rysujemy wielokąt, informacja o froncie karty zawiera liczbę odcinków do narysowania (kod implementujący rysowanie takiego wielokąta omawiamy w dalszej części). W przypadku drugiej wersji gry, w której rysujemy zdjęcia, wartością będzie referencja `img` do obiektu `Image` definiującego obraz do wyświetlenia. Oprócz obiektu obrazu będziemy przechowywać liczbę, która powiąże w parę różne obrazy. Do rysowania obrazu z pliku wykorzystamy standardową metodę `drawImage()`.

Oczywiście, w komputerze karty nie istnieją jako fizyczne obiekty, z dwoma stronami. Aplikacja rysuje na elemencie `canvas` tę stronę karty, która powinna być widoczna. Do rysowania koszulki służy funkcja `flipback()`. Usunięcie karty polega na narysowaniu w miejscu karty prostokąta w kolorze tła.

Obie aplikacje wykorzystują funkcję pod nazwą `makedeck()`, która „przygotowuje talię”, czyli generuje odpowiednie obiekty klasy `Card`. W przypadku pierwszej wersji w obiekcie `Card` zapisujemy liczbę wierzchołków wielokąta. Na tym etapie aplikacja jednak niczego nie rysuje. Wersja druga gry dodatkowo definiuje tablicę pod zmienną `pairs`, w której wymienione są nazwy plików fotografii. Wykorzystując tę tablicę, możesz podmienić zdjęcia fotografiami członków własnej rodziny lub przyjaciół.

Wskazówka: Jeśli chcesz wykorzystać moje przykładowe pliki, wystarczy, że pobierzesz z serwera FTP archiwa z przykładowymi kodami z tej książki. Jeśli zechcesz podmienić zdjęcia, musisz odpowiednio zmodyfikować kod aplikacji. W kodzie znajdują się informacje, w jaki sposób należy to zrobić.

Funkcja `makedeck()` tworzy obiekty klasy `Image` i wykorzystuje tablicę `pairs` do ustawienia ich właściwości `src`. Gdy kod tworzy obiekty klasy `Card`, ustawia ich atrybuty `index`, które definiują pary, to znaczy pasujące fotografie mają taką samą wartość `index`. Tak samo jak w wersji wykorzystującej wielokąt, aplikacja na etapie inicjalizacji nie rysuje niczego na elemencie `canvas`. Po narysowaniu karty wyglądają identycznie, ale powiązane z nimi informacje są zróżnicowane. Po zainicjalizowaniu karty są ułożone zawsze w tej samej kolejności, mieszanie następuje później.

Informacje o pozycji (właściwości `sx` i `sy`) są interpretowane w inny sposób w każdej z wersji aplikacji. W przypadku wersji wykorzystującej zdjęcia informacja odnosi się do lewego górnego wierzchołka karty. W przypadku wersji wykorzystującej wielokąt te współrzędne identyfikują środek wielokąta. Znając współrzędne środka, możemy jednak wyliczyć współrzędne wierzchołka i odwrotnie.

Wykorzystanie obiektu Date do obliczania czasu

Musimy zmierzyć czas, jaki graczowi zajęło znalezienie wszystkich dopasowań. JavaScript udostępnia sposób mierzenia czasu. Kompletny kod znajdziesz w punkcie „Zbuduj własną aplikację”, a w tym miejscu przedstawię jedynie fragmenty prezentujące sposób mierzenia czasu między dwoma zdarzeniami w kodzie JavaScript.

Wywołanie metody `Date()` generuje obiekt daty i czasu, np.:

```
starttime = new Date();
starttime = Number(starttime.getTime());
```

Powyższy fragment kodu zapisuje w zmiennej `starttime` liczbę milisekund (tysięcznych sekundy), jakie upłynęły od początku 1970 roku (w tym miejscu nie ma znaczenia, dlaczego czas jest mierzony od początku 1970 roku).

Gdy nasze gry „Pamięć” stwierdzą, że gracz znalazł wszystkie dopasowania, kreator `Date()` jest wywoływany ponownie:

```
var now = new Date();
var nt = Number(now.getTime());
var seconds = Math.floor(.5 + (nt - starttime) / 1000);
```

Oto analiza tego kodu:

1. Stworzenie nowego obiektu `Date` i przypisanie go zmiennej `now`.
2. Odczytanie czasu z obiektu `now` z użyciem metody `getTime()`. Wartość ta jest przekształcana na liczbę (funkcja `Number()`) i przypisywana zmiennej `nt`. To oznacza, że zmienna `nt` zawiera liczbę sekund, jakie upłynęły od początku 1970 roku do momentu wywołania metody `Date()`. Następnie program od tej wartości odejmuje zapisaną na początku gry zmienną `starttime`.
3. Dzielenie przez 1000 sprowadzi wartość do sekund.
4. Dodanie wartości `.5` spowoduje, że funkcja `Math.floor()` właściwie zaokrągli wartość do pełnych sekund.

Jeśli interesuje Cię większa dokładność pomiaru, możesz pominąć wyliczenie zaokrąglenia i dodanie wartości `.5`.

Takiego kodu możesz użyć wtedy, gdy chcesz zmierzyć czas wykonania fragmentów programu.

Obsługa pauzy

Gdy gramy w tego typu grę, używając prawdziwych kart, nie odczekujemy świadomie przed odwróceniem kart na drugą stronę. Ale, jak wspominałam wcześniej, implementacja komputerowa musi odczekać chwilę, żeby dać graczowi czas na obejrzenie obydwu odkrytych kart. Jak pamiętasz z rozdziałów 3. i 4. (animacja odbijającej się piłki oraz strzelanie z armaty i procy), do definiowania opóźnienia wykonania kodu w równych odstępach czasu wykorzystywaliśmy funkcję `setInterval()`. Do odczekania ustalonego czasu przed kontynuowaniem działania gry możemy wykorzystać inną funkcję o zbliżonym działaniu: `setTimeout()`. Pełny kod wykorzystujący tę funkcję znajdziesz

w podrozdziale „Zbuduj własną aplikację”. Zobaczmy, w jaki sposób definiuje się tego typu zdarzenie czasowe i co się stanie, gdy czas oczekiwania upłynie.

Funkcja `setTimeout()` ustawia pojedyncze zdarzenie czasowe, które jest wywoływane z ustalonym opóźnieniem. Funkcja `choose()`, wywoływana po tym, gdy gracz kliknie element `canvas`, sprawdza wartość zmiennej `firstpick` w celu określenia, czy kliknięta została pierwsza, czy druga karta z rzędu. W każdym z tych przypadków program rysuje front karty w tym samym miejscu, w którym znajdował się rysunek koszulki karty. Jeśli kliknięcie było drugim z rzędu i karty pasują do siebie, kod ustawia wartość zmiennej `matched` na `true` (jeśli karty nie pasują, wartość ta ustawiana jest na `false`). Jeśli gra nie jest jeszcze skończona, wywoływany jest następujący kod:

```
setTimeout(flipback, 1000);
```

To spowoduje, że po upływie jednej sekundy (1000 milisekund) zostanie wywołana funkcja `flipback()`. Funkcja `flipback()` wykorzystuje zmienną `matched`, która pozwala jej zdecydować, czy w miejscu odkrytych kart ma narysować ich koszulki (wartość `false`), czy usunąć karty ze stołu, rysując w ich miejscu kolor tła (wartość `true`).

Funkcji `setTimeout()` można użyć do definiowania indywidualnych zdarzeń czasowych. Wystarczy określić odstęp czasu oraz funkcję, która ma być wywołana z takim opóźnieniem. Pamiętaj jedynie, że jednostką czasu jest milisekunda.

Rysowanie tekstu

Standard HTML5 definiuje mechanizm rysowania tekstów na elemencie `canvas`. Dzięki temu mamy dostęp do bardziej dynamicznego, elastycznego sposobu wyświetlania tekstów niż dawniej. Łącząc tekst z prostokątami, liniami, łukami i obrazami, możemy uzyskać ciekawe efekty. W tym punkcie opiszę ogólne zasady posługiwania się tekstem w połączeniu z elementem `canvas`, przedstawię też prosty przykład do samodzielnego wypróbowania. Jeśli chcesz, możesz pominąć ten fragment i przejść do podrozdziału „Zbuduj własną aplikację”, w którym omawiam kod aplikacji prezentowanej na rysunkach od 5.5 do 5.8, czyli gry „Pamięć” z wykorzystaniem fotografii.

W celu umieszczenia tekstu na elemencie `canvas` napiszemy kod, który ustawia krój czcionki, a następnie wywołuje metodę `fillText()`, która rysuje tekst na elemencie `canvas` w ustalonym położeniu `x`, `y`. Poniższy przykład tworzy kilka napisów z wykorzystaniem różnych krojów czcionek (koniecznie przeczytaj uwagę w dalszej części tekstu).

```
<html>
  <head>
    <title>Czcionki</title>
    <script type="text/javascript">
      var ctx;
      function init(){
        ctx = document.getElementById('canvas').getContext('2d');
        ctx.font="15px Lucida Handwriting";
        ctx.fillText("this is Lucida Handwriting", 10, 20);
        ctx.font="italic 30px HarlemNights";
        ctx.fillText("italic HarlemNights",40,80);
        ctx.font="bold 40px HarlemNights"
        ctx.fillText("HarlemNights",100,200);
```

```

        ctx.font="30px Accent";
        ctx.fillText("Accent", 200,300);
    }
</script>
</head>
<body onLoad="init();">
    <canvas id="canvas" width="900" height="400">
        Twoja przeglądarka nie obsługuje elementu canvas standardu HTML5.
    </canvas>
</body>
</html>

```

Ten dokument uruchomiony w przeglądarce WWW daje efekt przedstawiony na rysunku 5.9.



Rysunek 5.9. Tekst z użyciem różnych czcionek wyświetlony na elemencie canvas z użyciem metod font() i fillText()

Ostrzeżenie: Pamiętaj, aby stosować czcionki dostępne w komputerach graczy. W rozdziale 10. dowiesz się, w jaki sposób wykorzystywać mechanizm standardu CSS pod nazwą font-family, który udostępnia systematyczny sposób definiowania preferowanych czcionek oraz czcionek zastępczych, jeśli podstawowa czcionka nie zostanie znaleziona.

Należy pamiętać, że mimo tego, iż tekst na elemencie canvas wygląda zupełnie „normalnie”, to należy go postrzegać tak jak druk na papierze albo obraz, a nie tekst, jaki znamy ze stron WWW, który można zaznaczać, lub pole tekstowe, którego wartość można modyfikować. Oznacza to, że w celu zmodyfikowania tekstu musimy napisać kod JavaScript, który wymaże starą zawartość. Wykorzystujemy do tego atrybut fillStyle, który ustawiamy na wartość zmiennej tablecolor, a następnie wywołujemy metodę fillRect() z odpowiednimi współrzędnymi.

Po narysowaniu ścieżki tekstu musimy wypełnić ją kolorem. Właściwość `fillStyle` musimy więc ustawić na inną wartość niż kolor tła (`tablecolor`). Możemy na przykład wykorzystać ten sam kolor, którego użyliśmy do namalowania koszulek kart. Krój czcionki ustawiamy w ramach inicjalizacji gry:

```
ctx.font = "bold 20pt sans-serif";
```

Użycie czcionki `sans-serif` ma sens, ponieważ krój ten należy do standardowych krojów internetowych, dostępnych na wszystkich komputerach.

Oto kod wypisujący liczbę dopasowań w danym momencie gry:

```
ctx.fillStyle = tablecolor;
ctx.fillRect(10, 340, 900, 100);
ctx.fillStyle=backcolor;
ctx.fillText("Liczba par: " + String(count), 10, 360);
```

Pierwsze dwie instrukcje wymazują aktualny wynik, a dwie kolejne wypisują aktualny. Wyrażenie `"Liczba par: " + String(count)` wymaga nieco więcej wyjaśnień. Jego funkcja jest następująca:

- Przekształca na łańcuch znaków wartość zmiennej `count`, która jest liczbą.
- Łączy ten łańcuch znaków z tekstem `"Liczba par: "`.

Łączenie łańcuchów znaków (konkatenacja) jest realizowane za pomocą operatora dodawania (plus), co demonstruje wieloznaczną interpretację tego operatora. Jeśli obydwa operandy są liczbami, znak plus powoduje dodawanie arytmetyczne. Jeśli operandy są łańcuchami znaków, operator plus powoduje połączenie tych łańcuchów w jeden (konkatenację). Sytuację, w której jeden operator ma wiele znaczeń, nazywa się **przeciążaniem operatorów**.

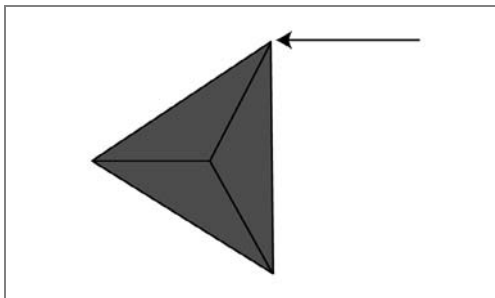
Co zrobi JavaScript, jeśli jeden z operatorów jest łańcuchem znaków, a drugi liczbą? Odpowiedź jest zależna od ich kolejności. W internecie można znaleźć mnóstwo przykładów, w których autorzy nie dokonali konwersji typów operandów, a konkatenacja znaków działa prawidłowo. Tajemnica leży we właściwej kolejności operacji.

Ja jednak sugeruję, żeby unikać niespodzianek i zawsze stosować operandy właściwych typów. Jeśli nagle okaże się, że program dodaje dwie jedynki i w efekcie otrzymujemy 11, a następnie 111, zamiast 1, 2, 3, to znak, że łączymy znaki, zamiast dodawać liczby i że należy dokonać konwersji typów.

Rysowanie wielokątów

Rysowanie wielokątów to dobra ilustracja możliwości HTML5 w zakresie rysowania. Aby zrozumieć proces tworzenia kodu używanego do rysowania wielokątów, warto wyobrazić sobie taki wielokąt jako kształt przypominający koło rowerowe ze szprychami rozchodzącymi się z jego środka do każdego z wierzchołków. Szprycha nie pojawia się na rysunku, ale służy jako pomoc w zrozumieniu procesu rysowania wielokąta. Koncepcję tę objaśnia rysunek 5.10, na przykładzie trójkąta.

W celu zmierzenia kąta między szprychami dzielimy wartość $2 * \text{Math.PI}$ (reprezentującą pełne koło) przez liczbę wierzchołków. Wartość kąta wykorzystamy do wywołania metody `moveTo()`, która przesuwa pozycję ścieżki w odpowiednie miejsce.



Rysunek 5.10. Reprezentacja trójkąta jako kształtu geometrycznego ze szprychami rozchodzącymi się z jego środka do wierzchołków. Strzałka wskazuje punkt, od którego zaczynamy rysowanie

Program rysuje wielokąt jako wypełnione ścieżki rozpoczynające się od punktu wskazanego strzałką na rysunku 5.10, wyznaczonego jako połowa wartości kąta między szprychami. Aby dotrzeć do tego punktu, wywołujemy metodę `moveTo()` z wartością promienia `radius` oraz wykorzystujemy funkcje `Math.sin()` i `Math.cos()`. Następnie metodę `lineTo()` wywołujemy $n - 1$ razy, postępując w kierunku zgodnym z ruchem wskazówek zegara. W przypadku trójkąta $n - 1$ to dwa kolejne wierzchołki. W przypadku ośmiokąta będzie to siedem kolejnych wierzchołków. Po zakończeniu pętli wywołujemy metodę `fill()`, która spowoduje wyświetlenie wypełnionego wielokąta. Kompletny kod z komentarzami znajdziesz w podrozdziale „Zbuduj własną aplikację”.

Uwaga: Rysowanie wielokątów zajmuje czas, ale w tej aplikacji to nie jest problem. Gdyby program wymagał rysowania dużej liczby skomplikowanych grafik, warto zastanowić się nad technikami przyspieszenia ich rysowania, jak na przykład użycie gotowych grafik. To rozwiązanie wymaga jednak pobierania dodatkowych plików, co z kolei również zajmuje czas. Należy przeprowadzić eksperymenty w celu stwierdzenia, które rozwiązanie jest najbardziej korzystne w danej sytuacji.

Mieszanie kart

Jak wspomniałam wcześniej, gra „Pamięć” wymaga tego, aby kolejność kart była różna przy każdej rozgrywce. Najlepszy sposób realizacji tego zadania wcale nie jest zagadnieniem trywialnym. W rozdziale 10. przy okazji gry w 21 znajdziesz odnośnik do artykułu omawiającego najefektywniejszy algorytm mieszania talii kart.

W przypadku gry „Pamięć” zaimplementujemy sposób, który wykorzystywałam, grając w tę grę jako dziecko. Razem z innymi graczami rozkładaliśmy karty, a następnie wybieraliśmy po dwie karty i zamienialiśmy je miejscami. Gdy uznaliśmy, że liczba zamian była wystarczająca, zaczęliśmy właściwą rozgrywkę. W tym punkcie omówię sposób zaimplementowania tego pomysłu w naszej aplikacji. Funkcję `shuffle()` znajdziesz w podrozdziale „Zbuduj własną aplikację”, w tabeli z pełnym kodem aplikacji.

W celu napisania kodu JavaScript niezbędnego do mieszania kart powinniśmy zdefiniować, co znaczy „wystarczająca liczba zamian”. Możemy przyjąć, że taką liczbą jest trzykrotna liczba kart w talii

zapisanych w tablicy `deck`. Ale przecież nie mamy fizycznych kart, a jedynie interpretujące je informacje. W takim razie co będziemy zamieniać? Oczywiście, tylko informacje identyfikujące kartę. W przypadku pierwszej wersji gry, wykorzystującej wielokąty, będzie to liczba boków, czyli właściwość `info`. W przypadku wersji wykorzystującej zdjęcia będą to właściwości `info` i `img`.

Do losowania kart wykorzystamy wyrażenie `Math.floor(Math.random() * d1)`, gdzie `d1` określa liczbę kart w talii. W każdym przebiegu takie losowanie wykonamy dwukrotnie, po czym zamienimy miejscami wylosowane dwie karty. Losowanie może dwukrotnie wskazać tę samą kartę, co oznacza, że zostanie ona zastąpiona przez siebie samą, ale nie jest to powód do zmartwienia. Jeśli coś takiego się zdarzy, po prostu jeden przebieg nie przyniesie zmian kolejności kart w talii. Zaplanowaliśmy dużą liczbę takich zamian, więc jedna mniej przejdzie niezauważona.

Kolejnym wyzwaniem jest realizacja samej zamiany. Do tej operacji potrzebne nam będą pomocnicze zmienne, przechowujące tymczasowo nasze przenoszone wartości. W przypadku wersji wykorzystującej wielokąty wykorzystamy jedną zmienną, pod nazwą `holder`, a w przypadku wersji ze zdjęciami będą to dwie zmienne: `holderimg` i `holderinfo`.

Kliknięcia w karty

Następnym krokiem będzie wyjaśnienie ruchów graczy, a dokładniej obsługa kliknięć kart. Obsługę zdarzeń kliknięć myszą rozwiążemy w sposób podobny do tego, jaki znamy z rozdziału 4. Wykorzystamy metodę `addEventListener()`:

```
canvas1 = document.getElementById('canvas');
canvas1.addEventListener('click', choose, false);
```

Ten fragment kodu umieścimy w funkcji `init()`. Funkcja `choose()` musi zawierać kod wykrywający klikniętą kartę. Program wykorzysta współrzędne kliknięcia myszą na elemencie `canvas`. Technikę odczytu współrzędnych również znamy z rozdziału 4.

Niestety, różne przeglądarki obsługują zdarzenia myszy na różne sposoby. Omówiłam to pokrótce w rozdziale 4., ale przypomnę te informacje również w tym miejscu. Poniższy kod działa prawidłowo w przeglądarkach Chrome, Firefox i Safari:

```
if ( ev.layerX || ev.layerX==0 ) {
    mx= ev.layerX;
    my = ev.layerY;
} else if (ev.offsetX || ev.offsetX==0 ) {
    mx = ev.offsetX;
    my = ev.offsetY;
}
```

Ten kod działa prawidłowo, ponieważ w przypadku gdy przeglądarka nie obsługuje właściwości `ev.layerX`, jej wartość będzie interpretowana jako `false`. Jeśli istnieje, ale ma wartość 0, również będzie interpretowana jako `false`, ale za to wyrażenie `ev.layerX==0` zwróci wartość `true`. Dzięki temu jeżeli program znajdzie użyteczną wartość `ev.layerX`, użyje jej. W przeciwnym razie kod sprawdzi istnienie właściwości `ev.offsetX`. Jeśli żadna z tych właściwości nie działa, zmienne `mx` i `my` nie zostaną przypisane.

Karty są prostokątami, więc łatwo będzie po prostu kolejno przejrzeć wszystkie karty i porównać ich współrzędne z pozycją myszy (mx , my) w momencie kliknięcia: wystarczy odczytać współrzędne lewego górnego wierzchołka oraz szerokość i wysokość karty. Warunek `if` skonstruujemy w następujący sposób:

```
if ((mx > card.sx) && (mx < card.sx + card.swidth) && (my > card.sy) && (my < card.sy + card.sheight)) {
```

Uwaga: W następnym rozdziale zajmiemy się sposobami dynamicznego definiowania kodu HTML z poziomu JavaScriptu. Poznamy w nim sposoby definiowania zdarzeń dla poszczególnych elementów na ekranie zamiast całego elementu canvas.

Wartość zmiennej `firstpick` ustawiamy na `true`, co sygnalizuje, że następne kliknięcie będzie pierwszym z dwóch w serii. Po kliknięciu pierwszej karty program zmieni tę wartość na `false`, a po kliknięciu drugiej ponownie na `true`. Tego typu zmienne, przełączane między dwoma stanami, często nazywa się **flagami** (ang. *flag*) lub **przełącznikami** (ang. *toggle*).

Zapobieganie oszustwom

Ten punkt omawia specyficzne zagadnienia dotyczące tej konkretnej gry, ale z tych przemyśleń wynika też ogólna nauka budowania interaktywnych aplikacji. Istnieją co najmniej dwa sposoby, w jakie gracz może próbować oszukać grę: klikając dwukrotnie na tej samej karcie lub klikając w miejscu, w którym znajdowała się wcześniej usunięta karta.

Przed pierwszym problemem zabezpieczymy się, umieszczając poniższy kod po warunku sprawdzającym, czy kliknięcie nastąpiło w obrębie karty:

```
if ((firstpick) || (i != firstcard)) break;
```

Ten wiersz kodu spowoduje wyjście z pętli `for`, jeśli indeks `i` jest prawidłowy, to znaczy jeśli kliknięta karta jest pierwszą w sekwencji lub drugą, ale inną od pierwszej.

Uniknięcie drugiego problemu (kliknięcia w miejscu, w którym już nie ma karty) wymaga więcej pracy. Gdy aplikacja usuwa kartę ze stołu, oprócz narysowania tła w miejscu karty może dodatkowo ustawiać specjalną wartość (np. `-1`) w jej właściwości `sx`. To będzie wirtualny znak, że karta jest usunięta. Operacja ta powinna być wykonana w funkcji `flipback()`. Funkcja `choose()` sprawdzi wartość właściwości `sx` (`sx >= 0`). Obydwa mechanizmy zapobiegania oszustwom można zaimplementować w następującej pętli `for`:

```
for (i = 0; i < deck.length; i++){
  var card = deck[i];
  if (card.sx >=0)
    if ((mx > card.sx) && (mx < card.sx + card.swidth) && (my > card.sy) && (my < card.sy + card.sheight)) {
      if ((firstpick)|| (i!=firstcard)) break;
    }
}
```

Mamy tu do czynienia z trzykrotnie zagnieżdżonymi instrukcjami warunkowymi. Druga z nich jest jedyną instrukcją pierwszej. Trzecia zawiera pojedyncze wyrażenie `break`, które powoduje wyjście z pętli. Z reguły zalecam stosowanie nawiasów klamrowych (`{ i }`) w instrukcjach warunkowych, ale powyższy przykład stanowi skróconą wersję, jedynie do celów demonstracyjnych.

Przejdźmy wreszcie do naszych dwóch wersji gry „Pamięć”.

Zbuduj własną aplikację

Ten podrozdział zawiera kompletne kody dwóch wersji gry „Pamięć”. Każda z nich wykorzystuje sporą liczbę funkcji, dlatego na początek zapoznaj się z ich zestawieniami w tabelach. Znajdziesz w nich nazwy funkcji oraz informacje, przez co są wywoływane i co same wywołują.

Tabela 5.1 zawiera zestawienie funkcji pierwszej wersji gry, wykorzystującej wielokąt. Niektóre z funkcji są wywoływane w ramach obsługi zdarzeń.

Tabela 5.1. Funkcje w wersji gry „Pamięć” wykorzystującej wielokąt

Funkcja	Wywoływana przez	Wywołuje
<code>init()</code>	obsługę zdarzenia <code>onLoad</code> elementu <code>body</code>	<code>makedeck()</code> <code>shuffle()</code>
<code>choose()</code>	obsługę zdarzenia myszy zdefiniowaną przez metodę <code>addEventListener()</code> w funkcji <code>init()</code>	<code>Polycard()</code> <code>drawpoly()</code> wywołaną jako metoda <code>draw()</code> wielokąta
<code>flipback()</code>	obsługę zdarzenia czasowego zdefiniowanego przez funkcję <code>setTimeout()</code> wywołaną w funkcji <code>choose()</code>	
<code>drawback()</code>	<code>makedeck()</code> i <code>flipback()</code> jako metodę <code>draw()</code> klasy <code>Polycard</code>	
<code>Polycard()</code>	<code>choose()</code>	
<code>shuffle()</code>	<code>init()</code>	
<code>makedeck()</code>	<code>init()</code>	
<code>Card()</code>	<code>makedeck()</code>	
<code>drawpoly()</code>	<code>choose()</code> jako metodę <code>draw()</code> klasy <code>Polygon</code>	

Tabela 5.2 zawiera kod pierwszej wersji gry z kompletnymi komentarzami. Analizując kod, zwróć uwagę na podobieństwa między tymi dwoma wersjami aplikacji, omówione wcześniej w tym rozdziale. Pamiętaj również, że to jest jeden z niezliczonej liczby sposobów zaprogramowania gry tego typu.

Tabela 5.2. Pełny kod pierwszej wersji gry „Pamięć” wykorzystującej wielokąt

<code><html></code>	Początek elementu <code>html</code>
<code><head></code>	Początek elementu <code>head</code>
<code><title>Gra "Pamięć"</title></code>	Kompletny element <code>title</code>

<code><style></code>	Początek elementu <code>style</code>
<code>form {</code>	Definicja stylu dla elementu <code>form</code>
<code>width:330px;</code>	Szerokość
<code>margin:20px;</code>	Margines zewnętrzny
<code>background-color:pink;</code>	Kolor
<code>Padding:20px;</code>	Wypełnienie wewnętrzne
<code>}</code>	Koniec definicji stylu
<code>input {</code>	Definicja stylu dla elementów pól formularza
<code>text-align:right;</code>	Wyrównanie do prawej, odpowiednie dla liczb
<code>}</code>	Koniec definicji stylu
<code></style></code>	Koniec elementu <code>style</code>
<code><script type="text/javascript"></code>	Początek elementu <code>script</code> . Deklaracja <code>type</code> nie jest niezbędna, ale zalecana
<code>var ctx;</code>	Zmienna przechowująca graficzny kontekst elementu <code>canvas</code>
<code>var firstpick = true;</code>	Deklaracja i inicjalizacja zmiennej <code>firstpick</code>
<code>var firstcard;</code>	Deklaracja zmiennej przechowującej informację o pierwszej karcie
<code>var secondcard;</code>	Deklaracja zmiennej przechowującej informację o drugiej karcie
<code>var frontbgcolor = "rgb(251,215,73)";</code>	Kolor tła frontów kart
<code>var polycolor = "rgb(254,11,0)";</code>	Kolor wielokątów
<code>var backcolor = "rgb(128,0,128)";</code>	Kolor koszulek kart
<code>var tablecolor = "rgb(255,255,255)";</code>	Kolor stołu
<code>var cardrad = 30;</code>	Promień wielokątów
<code>var deck = [];</code>	Deklaracja talii, początkowo pusta tablica
<code>var firstsx = 30;</code>	Współrzędna x pierwszej karty
<code>var firstsy = 50;</code>	Współrzędna y pierwszej karty
<code>var margin = 30;</code>	Odstęp między kartami
<code>var cardwidth = 4 * cardrad;</code>	Szerokość karty ustalona na czterokrotność promienia wielokąta
<code>var cardheight = 4 * cardrad;</code>	Wysokość karty ustalona na czterokrotność promienia wielokąta
<code>var matched;</code>	Ta zmienna jest ustawiana w funkcji <code>choose()</code> i wykorzystywana przez funkcję <code>flipback()</code>
<code>var starttime;</code>	Ta zmienna jest ustawiana w funkcji <code>init()</code> i służy do obliczenia czasu gry
<code>function Card(sx, sy, swidth, sheight, info) {</code>	Początek funkcji kreatora obiektu klasy <code>Card</code>
<code>this.sx = sx;</code>	Współrzędna pozioma
<code>this.sy = sy;</code>	... współrzędna pionowa
<code>this.swidth = swidth;</code>	... szerokość
<code>this.sheight = sheight;</code>	... wysokość

<code>this.info = info;</code>	... info (liczba boków wielokąta)
<code>this.draw = drawback;</code>	Metoda rysująca kartę
<code>}</code>	Koniec funkcji
<code>function madeck() {</code>	Początek funkcji <code>madeck()</code> inicjalizującej talie
<code>var i;</code>	Zmienna wykorzystywana w pętli <code>for</code>
<code>var acard;</code>	Zmienna przechowująca pierwszą kartę z pary
<code>var bcard;</code>	Zmienna przechowująca drugą kartę z pary
<code>var cx = firstsx;</code>	Zmienna przechowująca współrzędną x, inicjalizowana na współrzędną x pierwszej karty
<code>var cy = firstsy;</code>	Zmienna przechowująca współrzędną y, inicjalizowana na współrzędną y pierwszej karty
<code>for(i = 3; i < 9; i++) {</code>	Pętla rysująca wielokąty: od trójkątów po ośmiokąty
<code>acard = new Card(cx, cy, cardwidth, cardheight, i);</code>	Nowy obiekt karty
<code>deck.push(acard);</code>	Dodanie karty do talii
<code>bcard = new Card(cx, cy + cardheight + margin, cardwidth, cardheight, i);</code>	Druga karta z takimi samymi parametrami, ale wyświetlana poniżej pierwszej
<code>deck.push(bcard);</code>	Dodanie karty do talii
<code>cx = cx + cardwidth + margin;</code>	Odstęp między kartami z marginesem
<code>acard.draw();</code>	Rysowanie karty na elemencie <code>canvas</code>
<code>bcard.draw();</code>	Rysowanie karty na elemencie <code>canvas</code>
<code>}</code>	Koniec pętli <code>for</code>
<code>shuffle();</code>	Mieszanie kart
<code>}</code>	Koniec funkcji
<code>function shuffle() {</code>	Początek funkcji <code>shuffle()</code>
<code>var i;</code>	Zmienna przechowująca referencję do karty
<code>var k;</code>	Zmienna przechowująca referencję do karty
<code>var holder;</code>	Zmienna dodatkowa, niezbędna do dokonania zamiany
<code>var dl = deck.length</code>	Zmienna przechowująca liczbę kart w talii
<code>var nt;</code>	Indeks karty do zamiany
<code>for (nt = 0; nt < 3 * dl; nt++) {</code>	Pętla <code>for</code>
<code>i = Math.floor(Math.random() * dl);</code>	Pobranie losowej karty
<code>k = Math.floor(Math.random() * dl);</code>	Pobranie losowej karty
<code>holder = deck[i].info;</code>	Zapisanie informacji karty <code>i</code>
<code>deck[i].info = deck[k].info;</code>	Umieszczenie informacji karty <code>i</code> w miejscu karty <code>k</code>
<code>deck[k].info = holder;</code>	Umieszczenie poprzedniej informacji karty <code>k</code> w miejscu karty <code>i</code>
<code>}</code>	Koniec pętli <code>for</code>
<code>}</code>	Koniec funkcji

<code>function Polycard(sx, sy, rad, n) {</code>	Początek funkcji konstruktora klasy Polycard
<code> this.sx = sx;</code>	Ustawienie współrzędnej x
<code> this.sy = sy;</code>	... współrzędnej y
<code> this.rad = rad;</code>	... promienia wielokąta
<code> this.draw = drawpoly;</code>	... metody rysującej
<code> this.n = n;</code>	... liczby boków
<code> this.angle = (2 * Math.PI) / n</code>	Obliczenie i przypisanie kąta
<code>}</code>	Koniec funkcji
<code>function drawpoly() {</code>	Początek funkcji drawpoly()
<code> ctx.fillStyle = frontbgcolor;</code>	Wypełnienie kolorem frontu karty
<code> ctx.fillRect(this.sx - 2 * this.rad, this.sy - 2 * this.rad, 4 * this.rad, 4 * this.rad);</code>	Wierzchołek prostokąta znajduje się u góry i po lewej środka wielokąta
<code> ctx.beginPath();</code>	Początek ścieżki
<code> ctx.fillStyle = polycolor;</code>	Wypełnienie wielokąta
<code> var i;</code>	Zmienna używana do indeksowania
<code> var rad = this.rad;</code>	Promień
<code> ctx.moveTo(this.sx + rad * Math.cos(-.5 * this.angle), this.sy + rad * Math.sin(-.5 * this.angle));</code>	Przeniesienie rysowania do pierwszego wierzchołka
<code> for (i=1; i < this.n; i++) {</code>	Pętla for rysująca kolejne odcinki
<code> ctx.lineTo(this.sx + rad * Math.cos((i - .5) * this.angle), this.sy + rad * Math.sin((i - .5) * this.angle));</code>	Dodanie odcinka wielokąta do ścieżki
<code> }</code>	Koniec pętli for
<code> ctx.fill();</code>	Wypełnienie ścieżki
<code>}</code>	Koniec funkcji
<code>function drawback() {</code>	Początek funkcji
<code> ctx.fillStyle = backcolor;</code>	Ustawienie koloru koszulki karty
<code> ctx.fillRect(this.sx, this.sy, this.swidth, this.sheight);</code>	Rysowanie prostokąta
<code>}</code>	Koniec funkcji
<code>function choose(ev) {</code>	Początek funkcji choose() (obsługa kliknięcia karty)
<code> var mx;</code>	Zmienna przechowująca współrzędną x myszy
<code> var my;</code>	Zmienna przechowująca współrzędną y myszy
<code> var pick1;</code>	Zmienna przechowująca obiekt klasy Polygon
<code> var pick2;</code>	Zmienna przechowująca obiekt klasy Polygon
<code> if (ev.layerX ev.layerX == 0) {</code>	Czy przeglądarka obsługuje właściwości layerX i layerY?

<code>mx= ev.layerX;</code>	Ustawienie mx
<code>my = ev.layerY;</code>	Ustawienie my
<code>}</code>	Koniec warunku if true
<code>else if (ev.offsetX ev.offsetY == 0) {</code>	Czy przeglądarka obsługuje właściwości offsetX i offsetY?
<code>mx = ev.offsetX;</code>	Ustawienie mx
<code>my = ev.offsetY;</code>	Ustawienie my
<code>}</code>	Koniec klauzuli else
<code>var i;</code>	Zmienna wykorzystywana w pętli
<code>for (i = 0; i < deck.length; i++){</code>	Pętla przeglądająca wszystkie karty w talii
<code>var card = deck[i];</code>	Referencja do obiektu karty w celu uproszczenia kodu
<code>if (card.sx >= 0)</code>	Sprawdzenie, czy karta nie jest oznaczona jako usunięta
<code>if ((mx > card.sx) && (mx < card.sx + card.swidth) && (my > card.sy) && (my < card.sy + card.sheight)) {</code>	Sprawdzenie, czy kliknięcie nastąpiło w obrębie rysunku karty
<code>if ((firstpick) (i != firstcard)) break;</code>	Jeśli tak, sprawdzamy, czy to nie jest drugie kliknięcie w tę samą kartę; w takim przypadku kończymy przetwarzanie pętli
<code>}</code>	Koniec warunku if true
<code>}</code>	Koniec pętli for
<code>if (i < deck.length) {</code>	Czy nastąpiło wcześniejsze wyjście z pętli for?
<code>if (firstpick) {</code>	Jeśli to była pierwsza kliknięta karta...
<code>firstcard = i;</code>	... ustawiamy na nią referencję firstcard
<code>firstpick = false;</code>	Ustawienie firstpick na wartość false
<code>pick1 = new Polycard(card.sx + cardwidth * .5, card.sy + cardheight * .5, cardrad, card.info);</code>	Utworzenie obiektu wielokąta
<code>pick1.draw();</code>	Narysowanie wielokąta
<code>}</code>	Koniec warunku
<code>else {</code>	W przeciwnym wypadku...
<code>secondcard = i;</code>	... ustawienie referencji secondcard na drugą klikniętą kartę
<code>pick2 = new Polycard(card.sx + cardwidth * .5, card.sy + cardheight * .5, cardrad, card.info);</code>	Utworzenie obiektu wielokąta
<code>pick2.draw();</code>	Narysowanie wielokąta
<code>if (deck[i].info == deck[firstcard].info) {</code>	Sprawdzenie pary
<code>matched = true;</code>	Ustawienie zmiennej matched na wartość true
<code>var nm = 1 + Number(document.f.count.value);</code>	Inkrementacja liczby par

<code>document.f.count.value = String(nm);</code>	Wyświetlenie nowego wyniku
<code>if (nm >= .5 * deck.length) {</code>	Sprawdzenie, czy gra jest już zakończona
<code>var now = new Date();</code>	Odczyt obiektu daty i czasu
<code>var nt = Number(now.getTime());</code>	Wydobycie czasu i przekształcenie na liczbę
<code>var seconds = Math.floor(.5 + (nt - starttime) / 1000);</code>	Obliczenie sekund
<code>document.f.elapsed.value = String(seconds);</code>	Wyświetlenie czasu
<code>}</code>	Koniec warunku (czy koniec gry)
<code>}</code>	Koniec warunku (czy jest dopasowanie)
<code>else {</code>	w przeciwnym razie...
<code>matched = false;</code>	Ustawienie zmiennej <code>matched</code> na wartość <code>false</code>
<code>}</code>	Koniec instrukcji <code>else</code>
<code>firstpick = true;</code>	Przywrócenie domyślnego stanu zmiennej <code>firstpick</code>
<code>setTimeout(flipback, 1000);</code>	Ustawienie przerwy
<code>}</code>	Koniec warunku (jeśli to nie jest pierwsza karta)
<code>}</code>	Koniec warunku (kliknięcie w kartę, pętla <code>for</code> przerwana)
<code>}</code>	Koniec funkcji
<code>function flipback() {</code>	Początek funkcji <code>flipback()</code> obsługującej przerwę
<code>if (!matched) {</code>	Jeśli nie było dopasowania...
<code>deck[firstcard].draw();</code>	... rysowanie koszulki karty
<code>deck[secondcard].draw();</code>	... rysowanie koszulki karty
<code>}</code>	Koniec warunku
<code>else {</code>	Jeśli było dopasowanie (usuwanie kart)
<code>ctx.fillStyle = tablecolor;</code>	Ustawienie koloru wypełnienia na kolor stołu
<code>ctx.fillRect(deck[secondcard].sx, deck[secondcard].sy, deck[secondcard].swidth, deck[secondcard].sheight);</code>	Rysowanie w miejscu karty
<code>ctx.fillRect(deck[firstcard].sx, deck[firstcard].sy, deck[firstcard].swidth, deck[firstcard].sheight);</code>	Rysowanie w miejscu karty
<code>deck[secondcard].sx = -1;</code>	Ustawienie wartości powodującej pominięcie karty
<code>deck[firstcard].sx = -1;</code>	Ustawienie wartości powodującej pominięcie karty
<code>}</code>	Koniec warunku (brak dopasowania)
<code>}</code>	Koniec funkcji
<code>function init(){</code>	Początek funkcji <code>init()</code>

<code>ctx = document.getElementById('canvas').getContext('2d');</code>	Zmienna ctx realizująca rysowanie
<code>canvas1 = document.getElementById('canvas');</code>	Zmienna canvas1 do obsługi zdarzeń
<code>canvas1.addEventListener('click', choose, false);</code>	Ustawienie funkcji obsługi zdarzeń
<code>makedeck();</code>	Utworzenie talii
<code>document.f.count.value = "0";</code>	Inicjalizacja licznika par
<code>document.f.elapsed.value = "";</code>	Usunięcie starej wartości czasu
<code>starttime = new Date();</code>	Odczyt czasu rozpoczęcia gry
<code>starttime = Number(starttime.getTime());</code>	Ta sama zmienna jest użyta do zapisania właściwej wartości czasu w sekundach
<code>shuffle();</code>	Mieszanie kart
<code>}</code>	Koniec funkcji
<code></script></code>	Koniec elementu script
<code></head></code>	Koniec elementu head
<code><body onLoad="init();"></code>	Element body, konfiguracja uruchomienia funkcji init()
<code><canvas id="canvas" width="900" height="400"></code>	Początek elementu canvas
Twoja przeglądarka nie obsługuje elementu canvas standardu HTML5.	Komunikat dla użytkowników niekompatybilnych przeglądarek
<code></canvas></code>	Koniec elementu canvas
<code>
</code>	Przejdźcie do nowego wiersza
Klikaj karty próbując znaleźć pary.	Instrukcje gry
<code><form name="f"></code>	Początek elementu formularza
Liczba par: <code><input type="text" name="count" value="0" size="1"/></code>	Etykieta i pole tekstowe używane do wyświetlania informacji
<code><p></code>	Nowy akapit
Czas gry: <code><input type="text" name="elapsed" value=" " size="4"/></code> sekundy.	Etykieta i pole tekstowe używane do wyświetlania informacji
<code></form></code>	Koniec elementu form
<code></body></code>	Koniec elementu body
<code></html></code>	Koniec elementu html

Niezależnie od podjętych przez Ciebie decyzji programistycznych jeszcze raz zachęcam do komentowania kodu (używając dwóch ukośników // na początku) i robienia odstępów (pustych wierszy) między niezależnymi fragmentami kodu. Oczywiście, nie ma konieczności komentowania każdego wiersza, ale rozsądna liczba komentarzy z pewnością pomoże w przyszłości, gdy będziesz musiał wrócić do tego kodu w celu wprowadzenia poprawek lub udoskonaleń.

Grę można zmodyfikować, zmieniając rozmiar i kolor czcionki formularza lub kolor tła pola formularza. Więcej pomysłów na zmodyfikowanie aplikacji znajdziesz w dalszej części rozdziału.

Druga wersja gry „Pamięć”, wykorzystująca zdjęcia, posiada bardzo podobną strukturę do wersji wykorzystującej wielokąty. Wersja druga nie wymaga osobnej funkcji do rysowania frontu karty. Tabela 5.3 zawiera listę funkcji wykorzystywanych przez drugą wersję gry.

Tabela 5.3. Funkcje w drugiej wersji gry „Pamięć”, wykorzystującej fotografie

Funkcja	Wywoływana przez	Wywołuje
init()	obsługę zdarzenia onLoad elementu body	makedeck() shuffle()
choose()	obsługę zdarzenia myszy zdefiniowaną przez metodę addEventListener() w funkcji init()	
flipback()	obsługę zdarzenia czasowego zdefiniowanego przez funkcję setTimeout() wywołaną w funkcji choose()	
drawback()	makedeck() i flipback() jako metodę draw() klasy Polycard	
shuffle()	init()	
makedeck()	init()	
Card()	makedeck()	

Kod drugiej wersji aplikacji jest bardzo podobny do pierwszej. Większość logiki pozostaje bez zmian. Ta wersja do wyświetlania komunikatów do użytkownika korzysta z elementu canvas, dlatego dokument HTML nie zawiera formularza. Kod aplikacji jest przedstawiony w tabeli 5.4, z komentarzami tylko w miejscach różnic. W komentarzu sygnalizuję miejsce, w którym można zmodyfikować nazwy plików fotografii, jeśli zechcesz użyć własnych. Zanim zagłębisz się w analizę kodu drugiej wersji gry, zastanów się chwilę, które jej fragmenty będą się różnić od pierwszej, a które będą takie same.

Tabela 5.4. Kompletny kod drugiej wersji gry „Pamięć”, wykorzystującej fotografie

<html>	
<head>	
<title>Pamięć</title>	Kompletny element title
<script type="text/javascript">	
var ctx;	
var firstpick = true;	
var firstcard = -1;	
var secondcard;	
var backcolor = "rgb(128,0,128)";	
var tablecolor = "rgb(255,255,255)";	
var deck = [];	
var firstsx = 30;	

<code>var firstsy = 50;</code>	
<code>var margin = 30;</code>	
<code>var cardwidth = 100;</code>	Jeśli zdjęcia mają inne wymiary, w tym miejscu ustaw szerokość...
<code>var cardheight = 100;</code>	... a w tym wysokość
<code>var matched;</code>	
<code>var starttime;</code>	
<code>var count = 0;</code>	Licznik
<code>var pairs = [</code>	Tablica par zdjęć pięciu osób
<code>["allison1.jpg", "allison2.jpg"],</code>	W tej tablicy zdefiniuj nazwy plików ze zdjęciami
<code>["grant1.jpg", "grant2.jpg"],</code>	...
<code>["liam1.jpg", "liam2.jpg"],</code>	...
<code>["aviva1.jpg", "aviva2.jpg"],</code>	...
<code>["daniel1.jpg", "daniel2.jpg"]</code>	Możesz użyć dowolnej liczby par, ale ostatnia para nie może mieć przecinka po nawiasie kwadratowym
<code>]</code>	
<code>function Card(sx, sy, swidth, sheight, img, info) {</code>	
<code> this.sx = sx;</code>	
<code> this.sy = sy;</code>	
<code> this.swidth = swidth;</code>	
<code> this.sheight = sheight;</code>	
<code> this.info = info;</code>	Definicja par
<code> this.img = img;</code>	Referencja do obrazu
<code> this.draw = drawback;</code>	
<code>}</code>	
<code>function makedeck() {</code>	
<code> var i;</code>	
<code> var acard;</code>	
<code> var bcard;</code>	
<code> var pica;</code>	
<code> var picb;</code>	
<code> var cx = firstsx;</code>	
<code> var cy = firstsy;</code>	
<code> for(i = 0; i < pairs.length; i++) {</code>	
<code> pica = new Image();</code>	Utworzenie obiektu klasy Image
<code> pica.src = pairs[i][0];</code>	Ustawienie pierwszego pliku

ROZDZIAŁ 5.

<code>acard = new Card(cx, cy, cardwidth, cardheight, pic, i);</code>	Utworzenie karty (obiektu klasy Card)
<code>deck.push(acard);</code>	
<code>picb = new Image();</code>	Utworzenie obiektu obrazu (Image)
<code>picb.src = pairs[i][1];</code>	Ustawienie drugiego pliku
<code>bcard = new Card(cx, cy + cardheight + margin, cardwidth, cardheight, picb, i);</code>	Utworzenie karty (obiektu klasy Card)
<code>deck.push(bcard);</code>	
<code>cx = cx + cardwidth + margin;</code>	
<code>acard.draw();</code>	
<code>bcard.draw();</code>	
<code>}</code>	
<code>}</code>	
<code>function shuffle() {</code>	
<code>var i;</code>	
<code>var k;</code>	
<code>var holderinfo;</code>	Tymczasowa zmienna używana do zamiany
<code>var holderimg;</code>	Tymczasowa zmienna używana do zamiany
<code>var dl = deck.length</code>	
<code>var nt;</code>	
<code>for (nt = 0; nt < 3 * dl; nt++) {</code>	
<code> i = Math.floor(Math.random()*dl);</code>	
<code> k = Math.floor(Math.random()*dl);</code>	
<code> holderinfo = deck[i].info;</code>	Zapisanie wartości info
<code> holderimg = deck[i].img;</code>	Zapisanie wartości img
<code> deck[i].info = deck[k].info;</code>	Wpisanie wartości info z k do i
<code> deck[i].img = deck[k].img;</code>	Wpisanie wartości img z k do i
<code> deck[k].info = holderinfo;</code>	Wpisanie oryginalnej wartości info
<code> deck[k].img = holderimg;</code>	Wpisanie oryginalnej wartości img
<code>}</code>	
<code>}</code>	
<code>function drawback() {</code>	
<code> ctx.fillStyle = backcolor;</code>	
<code> ctx.fillRect(this.sx, this.sy, this.swidth, this.sheight);</code>	
<code>}</code>	
<code>function choose(ev) {</code>	

var out;	
var mx;	
var my;	
var pick1;	
var pick2;	
if (ev.layerX ev.layerX == 0) {	To jest kod obsługujący różnice między przeglądarkami
mx= ev.layerX;	
my = ev.layerY;	
} else if (ev.offsetX ev.offsetX == 0) {	
mx = ev.offsetX;	
my = ev.offsetY;	
}	
var i;	
for (i = 0; i < deck.length; i++){	
var card = deck[i];	
if (card.sx >= 0)	
if ((mx > card.sx) && (mx < card.sx + card.swidth) && (my > card.sy) && (my < card.sy + card.sheight)) {	
if ((firstpick) (i != firstcard)) {	
break;}	
}	
}	
if (i < deck.length) {	
if (firstpick) {	
firstcard = i;	
firstpick = false;	
ctx.drawImage(card.img, card.sx, card.sy, card.swidth, card.sheight);	Rysowanie zdjęcia
}	
else {	
secondcard = i;	
ctx.drawImage(card.img, card.sx, card.sy, card.swidth, card.sheight);	Rysowanie zdjęcia
if (card.info==deck[firstcard].info) {	Czy jest dopasowanie?
matched = true;	
count++;	Zwiększenie licznika
ctx.fillStyle = tablecolor;	

<code>ctx.fillRect(10, 340, 900, 100);</code>	Wymazanie w miejscu tekstu
<code>ctx.fillStyle = backcolor;</code>	Ustawienie koloru tekstu
<code>ctx.fillText("Liczba par: " + String(count), 10, 360);</code>	Wypisanie licznika
<code>if (count >= .5 * deck.length) {</code>	
<code>var now = new Date();</code>	
<code>var nt = Number(now.getTime());</code>	
<code>var seconds = Math.floor(.5 + (nt - starttime) / 1000);</code>	
<code>ctx.fillStyle = tablecolor;</code>	
<code>ctx.fillRect(0, 0, 900, 400);</code>	Wymazanie całej zawartości canvas
<code>ctx.fillStyle = backcolor;</code>	Kolor wypełnienia
<code>out="Czas gry: " + String(seconds) + " sekund.";</code>	Przygotowanie tekstu
<code>ctx.fillText(out, 10, 100);</code>	Wypisanie tekstu
<code>ctx.fillText("Odśwież stronę.", 10, 300);</code>	Wypisanie tekstu
<code>}</code>	
<code>}</code>	
<code>else {</code>	
<code>matched = false;</code>	
<code>}</code>	
<code>firstpick = true;</code>	
<code>setTimeout(flipback, 1000);</code>	
<code>}</code>	
<code>}</code>	
<code>}</code>	
<code>function flipback() {</code>	
<code>var card;</code>	
<code>if (!matched) {</code>	
<code>deck[firstcard].draw();</code>	
<code>deck[secondcard].draw();</code>	
<code>}</code>	
<code>else {</code>	
<code>ctx.fillStyle = tablecolor;</code>	
<code>ctx.fillRect(deck[secondcard].sx,deck[secondcard].sy, deck[secondcard].swidth, deck[secondcard].sheight);</code>	
<code>ctx.fillRect(deck[firstcard].sx, deck[firstcard].sy, deck[firstcard].swidth, deck[firstcard].sheight);</code>	

deck[secondcard].sx = -1;	
deck[firstcard].sx = -1;	
}	
}	
function init(){	
ctx = document.getElementById('canvas'). getContext('2d');	
canvas1 = document.getElementById('canvas');	
canvas1.addEventListener('click', choose, false);	
makedeck();	
shuffle();	
ctx.font="bold 20pt sans-serif";	Ustawienie kroju czcionki
ctx.fillText("Klikaj karty, próbując znaleźć pary.", 10, 20);	Wypisanie instrukcji na elemencie canvas
ctx.fillText("Liczba kart: 0",10,360);	Wypisanie licznika
starttime = new Date();	
starttime = Number(starttime.getTime());	
}	
</script>	
</head>	
<body onLoad="init();">	
<canvas id="canvas" width="900" height="400">	
Twoja przeglądarka nie obsługuje elementu canvas standardu HTML5.	
</canvas>	
</body>	
</html>	

Obydwa przedstawione programy są kompletnymi grami, ale to nie znaczy, że nie da się ich udoskonalić. Na przykład gracz nie może przegrać. Warto zatem zastanowić się nad udostępnieniem takiej możliwości, na przykład przez ograniczenie liczby prób lub ustalenie limitu czasu.

Obie aplikacje uruchamiają zegar od razu po załadowaniu. Niektóre gry czekają z rozpoczęciem odmierzenia czasu do wykonania pierwszej czynności przez gracza. Jeśli chcesz zaimplementować tego typu podejście, zdefiniuj zmienną ustawioną pierwotnie na wartość `false`. Funkcja `choose()` po prostu sprawdzi jej wartość, a jeśli to `false`, ustawi ją na `true` oraz ustawi wartość zmiennej `starttime` na bieżący czas.

To jest gra dla jednego gracza. Można wymyślić sposób przystosowania jej dla dwóch graczy. Należałoby założyć, że osoby grają na zmianę, ale program śledzi ich wyniki w celu porównania.

Gracze lubią gry pozwalające na dobranie poziomu trudności. W tym celu można dostosować liczbę kart, skrócić czas wyświetlania odkrytych kart lub zastosować inne techniki.

Aplikacje możesz przystosować do własnych potrzeb, wykorzystując prywatne zdjęcia. Oczywiście, można użyć zdjęć przyjaciół i rodziny, ale można też stworzyć grę z elementami edukacyjnymi, wykorzystując logiczne pary ilustracji, jak symbole nut i ich nazwy, nazwy państw i ich stolic, mapy państw i ich nazwy itp. Można też zmienić liczbę par. Kod w swojej logice wykorzystuje długości tablic, nie ma więc potrzeby zmieniania wartości różnych zmiennych w kodzie w celu zmiany liczby kart w talii. Być może warto zmodyfikować szerokość i wysokość kart, aby zmieściły się na ekranie.

Inną możliwością jest użycie standardowej talii 52 kart (54 z jokerami). Przykład użycia prawdziwych kart znajdziesz w rozdziale 10., przy okazji gry w 21. W przypadku każdej gry w dopasowania warto zadbać o to, aby gracz miał jasną informację, czym dokładnie jest dopasowanie.

Testowanie aplikacji i wrzucenie jej na serwer

Gdy programiści testują swoje programy, z reguły robią rzeczy rozsądne. Użytkownicy, gracze i klienci mają jednak skłonność do robienia rzeczy nielogicznych. Dlatego zawsze warto poprosić osobę postronną o przetestowanie aplikacji. Poproś przyjaciół, żeby przetestowali Twoją grę. Zawsze warto zadbać o testerów, którzy nie brali udziału w pisaniu aplikacji.

Dokument HTML w wersji wykorzystującej wielokątę zawiera kompletną grę, ponieważ wszelkie grafiki są rysowane bezpośrednio w kodzie. Wersja gry wykorzystująca fotografie wymaga przesłania na serwer również wszystkich plików. Grę można urozmaicić przez użycie obrazów z zewnętrznych stron WWW (spoza Twojego serwera WWW). W takim przypadku w tablicy `pairs` należy podać pełne adresy URL obrazów.

Podsumowanie

W tym rozdziale nauczyłeś się implementować dwie wersje znanej gry „Pamięć” (znanej również jako „Koncentracja”), wykorzystując do tego celu funkcje standardów HTML5 i JavaScriptu, między innymi:

- funkcje i obiekty definiowane przez programistę;
- rysowanie wielokątów na elemencie `canvas` z użyciem metod `moveTo()` i `lineTo()` w połączeniu z funkcjami trygonometrycznymi zdefiniowanymi w module `Math`;
- wykorzystanie formularza do wyświetlania komunikatów dla użytkownika;
- rysowanie tekstu na elemencie `canvas` z użyciem wybranego kroju czcionki;
- rysowanie obrazów na elemencie `canvas`;
- wykorzystanie funkcji `setTimeout()` do wstrzymania działania programu;
- wykorzystanie obiektów `Date` do odmierzenia upływu czasu.

Przy okazji implementacji znanej gry miałeś okazję poznać sposoby reprezentowania informacji. Następny rozdział na chwilę odejdzie od elementu `canvas`. Poznamy sposoby dynamicznego generowania i pozycjonowania elementów HTML. Nauczymy się też wykorzystywać element `video`, wprowadzony w HTML5.

Skorowidz

A

adres
 bezwzględny, 31, 194
 lokalny, 32
 względny, 31

akapit, 22

algorytm Fishera-Yatesa, 294

anchor, kotwica, 19

animacja, 76

animowane obrazy GIF, 76

aplikacja Flash, 76

argumenty, 41

arkusz stylów CSS, 15, 22, 177, 268

ASCII, 71

aspect ratio, proporcje obrazu, 80

atribut, 19
 autoplay, 181, 248
 checked, 214
 class, 24, 178
 codecs, 181
 controls, 181, 248
 fillStyle, 84, 148
 font-size, 24
 height, 19
 href, hypertext reference, 19
 id, 24, 45, 176
 img, 238
 innerHTML, 177
 layerX, 240
 layerY, 240
 loop, 248
 margin, 24
 name, 53
 offsetX, 240
 offsetY, 240
 onClick, 209
 onLoad, 42
 padding, 24
 pattern, 89

picture, 238
preload, 181
rectcolor, 238
src, 19, 72, 181, 248
style, 22
text-align, 24
textContent, 177
type, 181
value, 53
width, 19

B

baza danych typu klucz-wartość, 207

białe znaki, 18

błędy, 32
 przeglądarki, 210
 semantyki, 73

C

camel case, 40

ciasteczka, cookie, 199, 207

CSS, Cascading Style Sheets, 15, 22, 177, 268

czcionki, 148, 244

D

data i czas, 207

definicja
 konstruktora MCard(), 290
 stylu formularza, 243
 stylu klasy letters, 268
 stylu klasy blanks, 268
 stylu pola tekstowego, 243

definiowanie
 czcionek, 148
 elementów HTML, 175
 funkcji, 41

grubości linii, 47
 obiektu, 108
 stylów, 22

deklarowanie zmiennej, 40

detekcja kolizji, 87, 207

dodawanie elementów, 175

dostęp do obiektu, 79

dynamiczne
 definiowanie elementów
 HTML, 175
 tworzenie elementów HTML, 266

dziedziczenie, 244

E

efekt tęczy, 82

element
 a, 19
 article, 22, 296
 audio, 247
 body, 18, 45
 button, przycisk, 42, 53
 canvas, 16, 35, 45, 81
 div, 175, 268
 footer, 16, 22, 283, 296
 form, 42, 53
 head, 18, 22
 header, 16, 22, 283, 296
 html, 18
 img, 19, 72
 nav, 296
 p, 22
 script, 25
 section, 16, 22
 source, 181
 style, 22, 178, 268
 submit, 53
 title, 18
 video, 178, 180, 182

element potomny elementu body, 177

F

filmy video, 172
 flaga, flag, 152
 Flash, 76
 format tekstowy, 18
 formularz, 53, 71, 179
 funkcja
 addEventListener(), 137, 215, 240
 alert(), 208
 builddeck(), 289, 291
 Card(), 145
 choose(), 147, 151, 240
 clearInterval(), 86, 257
 Date(), 25, 212
 deal(), 291
 dealfromdeck(), 291, 292, 293
 dealstart(), 291
 distsq(), 116
 drawall(), 115
 drawball(), 108
 drawface(), 55
 drawnoose(), 270
 drawsling(), 113
 drawThrow(), 238
 findball(), 115
 finish(), 115, 202
 fire(), 116
 flipback(), 145, 147, 152
 flyin(), 245
 getKey(), 295
 getKeyAndMove(), 203
 getwalls(), 214
 Image(), 79
 init(), 42, 55, 85, 202
 intersect(), 204
 isNumber(), 88
 makedeck(), 145
 Math.cos(), 150
 Math.floor(), 146
 Math.sin(), 150
 MCard(), 290
 more_to_house(), 292, 293
 moveandcheck(), 87
 moveball(), 85, 86, 108
 moveit(), 115
 newgame(), 294
 Number(), 71

picklement(), 177, 183, 269, 271
 setInterval(), 84, 86, 137
 setTimeout(), 84, 147, 166, 188
 setupgame(), 183, 267
 shuffle(), 150, 294
 startwall(), 202
 store(), 209
 String(), 71
 Throw(), 238
 throwdice(), 42, 72
 Token(), 202
 typeof(), 209
 funkcje
 konstruktora, 108, 237
 obsługi zdarzenia, event handler, 85
 przywiązane do elementu HTML, 42
 rysujące, 271
 wywoływanie, 42
 funkcje w aplikacji
 „Blackjack”, 297
 „Kamień, papier, nożyce”, 250
 „Kula armatnia”, 117
 „Labirynt”, 215
 „Odbijająca się piłka”, 90
 „Pamięć”, 153, 160
 „Proca”, 129
 „Przejdź labirynt”, 224
 „Quiz”, 183
 „Rzut dwiema kośćmi”, 61
 „Rzut pojedynczą kością”, 55
 „Strzał z armaty”, 122
 „Wiselec”, 273

G

generowanie animacji, 236
 gra, 17
 blackjack, 283, 297
 craps, 42
 kamień, papier, nożyce, 233
 labirynt, 195, 215
 quiz, 167, 183
 w kości, 35, 66
 w pamięć, 139
 wiselec, 259, 266
 gradient, 81
 GUI, graficzny interfejs użytkownika, 38

H

HTML, Hypertext Markup Language, 15

I

I/O, 38
 identyfikator #vid, 182
 indeks, 81
 instrukcja
 break, 44, 214
 case, 44
 if, 42
 new, 79
 switch, 44, 203
 var, 40, 79
 instrukcje try...catch, 210

J

jakość aplikacji, 247
 JavaScript, 24
 język
 interpretowany, 196
 kompilowany, 24, 196
 skryptowy, 24

K

catalog, 32
 Kerr Cheridan, 10
 klasa
 Card, 145
 Image, 79
 Math, 38
 thing, 178
 Throw, 238
 Wall, 202
 klucz, 207
 kod aplikacji
 „Blackjack”, 298
 „Kamień, papier, nożyce”, 250
 „Kula armatnia”, 118
 „Labirynt”, 215, 224
 „Odbijająca się piłka”, 90
 „Pamięć”, 153, 160
 „Proca”, 130
 „Quiz”, 183, 189

kod aplikacji

- „Rzut dwiema kośćmi”, 61
- „Rzut pojedynczą kością”, 56
- „Strzał z armaty”, 123
- „Wisielec”, 274
- gra w kości, 66
- zapisującej datę i czas, 210

kod

- CSS, 18
- do wywołania, 78
- dokumentu „Moje gry”, 28
- dokumentu „Ulubione strony”, 30
- dynamicznie tworzący elementy HTML, 266
- funkcji drawThrow(), 238
- HTML, 18
- HTML generowany dynamicznie, 174
- JavaScript, 18
- przechowujący wynik, 71
- wykrywania kolizji, 87

kody klawiszy, 204

koercja obliczeń, 206

kolizja, 128

kolor tekstu, 23

kolory zdefiniowane, 179

komentarze, 60, 204

konkatenacja, *Patrz* łączenie łańcuchów

konstruktor, 108

- Ball(), 108
- MCard(), 290
- String, 245
- Token(), 202

kontekst graficzny, 46

kontrola odtwarzania, 180

konwerter Miro, 247

kotwica, anchor, 19

L, Ł

liczby pseudolosowe, 35

lista odnośników, 16

lokalny magazyn danych, local storage, 196, 199

losowanie kart, 151

losowanie par liczb, 174

łączenie łańcuchów, 84, 149

M

magazyn

- localStorage, 209, 212
- lokalny, 196, 199, 207

marketing zachowaniowy, behavioral marketing, 199

metoda, 25

- addEventListener(), 176, 194, 203, 295
- arc(), 48
- ctx.beginPath(), 47
- ctx.clearRect(), 52
- ctx.closePath(), 47
- ctx.fill(), 48, 51
- ctx.lineTo(), 47
- ctx.moveTo(), 47, 269
- ctx.restore(), 110, 270
- ctx.save(), 110, 270
- ctx.scale(), 270
- ctx.stroke(), 47
- Date(), 146
- document.getElementsByTagName(), 249
- document.body.appendChild(), 194
- document.createElement(), 194
- document.getElementById(), 194, 249
- metoda document.write(), 25, 85
- draw(), 107
- drawImage(), 80, 145
- fill(), 113
- fillRect(), 71, 148
- fillStyle(), 51, 71, 81
- fillText(), 147, 148
- font(), 148
- getElementByTagName(), 257
- getTime(), 146
- join(), 213
- lineTo(), 114
- localStorage.setItem(), 213
- Math.atan2(), 116
- Math.floor(), 39
- Math.random(), 38, 174, 241
- Math.sqrt(), 206
- moveit(), 108
- moveTo(), 114, 150

play(), 182

preventDefault(), 203, 295

push(), 109, 137

removeEventListener(), 269

rotate(), 111

splice(), 116, 137, 183

split(), 213

stroke(), 50, 113, 269

strokeStyle(), 51

substr(), 176

substring(), 176

write(), 25

Meyer Jeanine, 9

modyfikacja w czasie działania, runtime, 175

modyfikowanie tablic, 137

N

nagłówek, 22

nazwa funkcji, 41

nazwy zmiennych, 40

notacja, 26

O

obiekt

Card, 145

document, 25

htarget, 117

localStorage, 209, 212

window, 203

obiekty

klas wbudowanych, 107

współdzielone, shared objects, 207

obliczanie prędkości, 113

obrót, 110

obrys, stroke, 113

obsługa

banku gracza, 72

błędów, 210, 213

dźwięku, 247, 256

formularzy, 53

gra w kości, 53

klawiatury, 295

klawiszy strzałek, 203

kontekstu graficznego, 16

paazy, 146

reguł gry, 65
 zdarzenia key, 203
 zdarzeń, 215, 240
 zdarzeń myszy, 137, 202
 odległość między punktami, 117
 odnośnik, 19
 odświeżanie strony, 32
 okno przeglądarki, 46
 operacje wejścia-wyjścia (I/O), 38
 operator
 +=, 83
 inkrementacji ++, 83
 new, 108
 przypisania, 43

P

pamięć podręczna przeglądarki, 89
 para klucz-wartość, 212
 pętla
 do...while, 175
 for, 83, 173
 while, 292
 plik sword.mp3, 248
 pliki
 dźwiękowe, 248
 htaccess, 194
 HTML, 27
 ilustracji, 19
 źródłowe, 248
 pole video, 180
 postinkrementacja, 290
 pozycjonowanie bezwzględne,
 absolute, 178
 pozycjonowanie względne,
 relative, 179
 program, 196
 Miro, 247
 TextPad, 26
 TextWrangler, 26
 program graficzny
 Adobe Flash Professional, 23
 Adobe Photoshop, 23
 Corel Paint Shop Pro, 23
 programowanie zdarzeniowe, 203
 programowanie zorientowane
 obiektowo, 117
 proporcje obrazu, aspect ratio, 80
 prosta parametryczna, 205

przechowywanie stanu gry, 66
 przechwytywanie klawiszy, 295
 przeciążanie operatorów, 149, 244
 przecięcie okręgu z odcinkiem, 205
 przeglądarka
 Chrome, 32
 Firefox, 32
 Safari, 32
 przełącznik, toggle, 152
 przyciski alfabetu, 267
 przypisywanie wartości
 zmiennym, 40
 pseudokod, 43
 punkt początkowy, origin, 46
 punkty zmiany koloru, 81

R

radian, 110
 RGB, red green blue, 23
 rozmiar pola, 245
 rozmiar tablicy, 174
 rysowanie, 45
 głowy, 270
 gradientu, 81, 93
 linii, 113
 na elemencie canvas, 78
 nowego obrazu, 52
 obrazu z pliku, 145
 odcinków, 113
 okręgu, 270
 piłki, 78
 procy, 113
 prostokąta, 46
 przycisków, 239
 ramki, 79
 szubienicy, 269
 ściany, 202
 ścieżki, 47
 tekstu, 147
 wielokątów, 149
 rzutowanie, 245

S

selektor, 268
 selektor elementów, 22
 singleton tag, 19
 składnia, 26
 skrypt, 196

słowo kluczowe
 function, 41
 return, 41
 specyfikacja HTML5, 24
 sterowanie grą, 236
 struktura dokumentu
 artykuł, 22
 nagłówek, 22
 sekcja, 22
 stopka, 22
 symulacja grawitacji, 106
 szpiegowanie użytkowników, 199

Ś

ścieżka, path, 19, 47
 śledzenie stanu, 38
 śledzenie stanu gry, 271, 282

T

tablica
 beats, 242
 chunks, 175
 everything, 81, 122
 facts, 174, 175
 pairs, 166, 173
 points, 242
 slots, 175
 steps, 270
 tablic, 82, 194
 walls, 205, 214
 words, 266
 tablice wewnętrzne, 82
 testowanie aplikacji, 32, 60, 72, 100,
 166, 193, 231, 257, 282, 306
 trzeci wymiar stron WWW, 181
 tworzenie
 nowych obiektów, 79
 przycisków, 237
 rysunku na elemencie canvas, 269
 tytuł dokumentu, 18

U

UNICODE, 71
 URL, Universal Resource Locator, 20
 usuwanie funkcji obsługi zdarzenia,
 269

W

walidacja, 53, 88
walidacja formularza, 97
wartości boolowskie
 false, 43
 true, 42
wartość klucza, 209
właściwość
 font-family, 257, 269
 visibility, 181
wypełnienie, fill, 113
wyrażenie, 39
 prices.length, 174
 this.textContent, 272
wyświetlanie
 czasu, 85
 filmów, 167
 wideo, 180
wywołanie
 closePath(), 51
 funkcji, 42
 JavaScript, 46

Z

zagnieżdżanie elementów, 18
zapisywanie
 danych, 202, 230
 daty i czasu, 207
zawijanie wierszy, 28
zdarzenia myszy, 151
zdarzenie
 keydown, 295
 keypress, 295
 keyup, 295

zegar, 165
z-index, 181, 182
zmienna, 40
 ballrad, 87
 ballx, 88
 chicken, 116
 class, 176
 ctx, 45, 80
 curwall, 202
 family, 81
 feathers, 116
 firstturn, 66
 grad, 81
 gravity, 136
 inmotion, 114, 202
 last, 209
 lettersguessed, 281
 mon, 44
 olddate, 208
 pairs, 145
 PI, 48
 secs, 86
 starttime, 165
 swalls, 213
 tablecolor, 148
 tev, 86
 this, 108, 177, 237
 unmotion, 114
zmiennie
 globalne, 41
 indeksujące, 289
 lokalne, 41
znacznik, 18
 !DOCTYPE, 19
 końcowy, 18

 początkowy, 18
 pojedynczy, 19
 input, 53

, 21
 img, 19
znaczniki od h1 do h6, 22
znak
 cudzysłowu, 43
 dodawania, 84
 kropki, 25
 krzyżyka, 268
 łącznika, 179, 182
 przejścia do nowego
 wiersza, 21
 równości, 40
 średnika, 25
 większości, 43

Ź

źródło ilustracji, 289
źródło, source, 19

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ 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>

Wprowadzenie do HTML5. Nauka HTML5 i JavaScriptu na przykładzie gier

Nauka jeszcze nigdy nie była tak wciągająca!

Standard HTML5 umożliwia tworzenie nie tylko dynamicznych, multimedialnych stron WWW, ale także zaawansowanych aplikacji internetowych i gier. Do niedawna, żeby to osiągnąć, konieczna była instalacja dodatkowych rozszerzeń. Teraz można to zrealizować, korzystając wyłącznie z HTML-a, JavaScriptu i kaskadowych arkuszy stylów CSS... Jeśli chcesz szybko i przyjemnie poznać podstawowe założenia HTML5 oraz natychmiast wykorzystać je w praktyce, sięgnij po tę książkę. W charakterze materiału do analizy użyto w niej popularnych gier, takich wisielec, kości czy blackjack.

Wprowadzenie do HTML5 pokaże Ci nowe, ekscytujące możliwości standardu HTML5 oraz sposoby integrowania HTML-a z JavaScriptem i CSS. Dzięki podanym przykładom zrozumiesz, jak te elementy wzajemnie się uzupełniają. Każdy rozdział jest poświęcony innej aplikacji i ma podobną strukturę. Na początku znajdują się wymagania, a następnie funkcje HTML5, CSS i JavaScriptu niezbędne do konkretnej implementacji. W dalszej części szczegółowo omawiana jest sama implementacja. Jeśli masz nieco większe doświadczenie w tworzeniu stron WWW, ta książka przybliży Ci sposoby wykorzystania JavaScriptu w połączeniu z dokumentami HTML oraz zapozna Cię z nowościami w HTML5, takimi jak obsługa zdarzeń, walidacja formularzy, wykorzystanie lokalnego magazynu danych czy elementy canvas, wideo i audio. Przykłady są wystarczająco proste, aby ich zrozumienie nie przysparzało problemów, ale na tyle inspirujące, by dać możliwość stworzenia ciekawych interaktywnych aplikacji WWW.

Połącz przyjemne z pożytecznym:

- Poznaj JavaScript, CSS i HTML5
- Naucz się rysować w przeglądarce internetowej
- Korzystaj ze zdarzeń
- Przechowuj dane w przeglądarce użytkownika
- Bądź na bieżąco z najnowszymi trendami



helion.pl
księgarnia
internetowa

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nawosci>



Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Informatyka w najlepszym wydaniu

