

Jacob Seidelin



HTML5

TWORZENIE GIER

Już dziś zacznij tworzyć gry w HTML5!

Tytuł oryginału: HTML5 Games: Creating Fun with HTML5, CSS3, and WebGL

Tłumaczenie: Maciej Reszotnik

Projekt okładki: Jan Paluch

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

ISBN: 978-83-246-4480-3

© 2012 John Wiley and Sons, Ltd.

All Rights Reserved. Authorised translation from the English language edition published by John Wiley & Sons Limited. Responsibility for the accuracy of the translation rests solely with Helion S.A. and is not the responsibility of John Wiley & Sons Limited.

No part of this book may be reproduced in any form without the written permission of the original copyright holder, John Wiley & Sons Limited.

Translation copyright © 2012 by Helion S.A.

Wiley and the John Wiley & Sons, Ltd logo are trademarks or registered trademarks of John Wiley and Sons, Ltd and/ or its affiliates in the United States and/or other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Ltd is not associated with any product or vendor mentioned in the book.

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

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

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

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą 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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/htm5tg>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem: <ftp://ftp.helion.pl/przyklady/htm5tg.zip>

Printed in Poland.

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

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

Spis treści

Wstęp	13
Część I. Wstęp do gier w standardzie HTML5	17
Rozdział 1. Gry w sieci	19
HTML5 — rys historyczny	20
HTML5 a gry	20
Element canvas	21
Dźwięk	22
WebSocket	23
Magazyn sieciowy	24
WebGL	24
HTML5 (nie) jest pogromcą Flasha	25
Gwarantowanie wstecznej kompatybilności	25
Wykrywanie własności	25
Używanie biblioteki Modernizr	26
Wypełnianie luk	26
Konstruowanie gry	27
Podsumowanie	28
Rozdział 2. Pierwsze kroki	29
Planowanie gry	30
Dopasowywanie brylantów	30
Dopasowywanie	30
Etapy gry	31
Definiowanie warstw gry	31
Ekran powitalny	31
Menu główne	31
Gra	32
Rekordowe wyniki	32
Tworzenie szkieletu aplikacji	33
Struktura HTML	34
Nadawanie stylu	34
Wczytywanie skryptów	36

Projektowanie ekranu powitalnego	40
Używanie fontów sieciowych	40
Formatowanie ekranu powitalnego	41
Podsumowanie	43
Rozdział 3. HTML i urządzenia mobilne	45
Tworzenie aplikacji sieciowych na urządzenia przenośne	46
Napisz raz a dobrze	46
Platformy mobilne — wyzwania	47
Obsługa sterowania na urządzeniu mobilnym	47
Sterowanie za pomocą klawiatury	47
Mysz kontra dotyk	48
Przystosowywanie gry do niskiej rozdzielczości	49
Projektowanie skalowalnych układów strony	50
Kontrolowanie przestrzeni operacyjnej	51
Wyłączenie skalowania	52
Tworzenie różnych widoków	52
Tworzenie głównego menu	53
Dodawanie modułów ekranu	54
CSS i zapytania medialne	57
Wykrywanie orientacji urządzenia	58
Dodawanie mobilnych arkuszy stylów	59
Tworzenie gier na systemy iOS i Android	61
Umieszczanie aplikacji na ekranie głównym telefonu	61
Usunięcie interfejsu przeglądarki	68
Debugowanie aplikacji mobilnych	71
Włączanie debugera Safari	71
Uzyskanie dostępu do logów w Androidzie	73
Podsumowanie	73
Część II. Tworzenie komponentów gry	75
Rozdział 4. Konstruowanie gry	77
Tworzenie modułu planszy	78
Inicjalizowanie stanu gry	79
Wypełnianie planszy	81
Implementacja zasad	83
Walidacja ruchów gracza	83
Wykrywanie łańcuchów	85
Generowanie nowych klejnotów	87
Przyznawanie punktów	88
Ponowne wypełnianie siatki	89
Dopasowywanie brylantów	92
Podsumowanie	93
Rozdział 5. Pracownicy i delegowanie zadań	95
Pracownicy sieciowi	96
Pracownicy — ograniczenia	96
Możliwości pracowników sieciowych	97

Korzystanie z pracowników	98
Wysyłanie wiadomości	98
Otrzymywanie wiadomości	99
Przechwytywanie błędów	99
Współdzieleni pracownicy	100
(Nie) Pierwszy przykład	101
Wykorzystywanie pracowników w grach	103
Tworzenie modułu pracownika	104
Utrzymanie starego interfejsu	105
Podsumowanie	110
Rozdział 6. Element canvas i grafika	111
Grafika w sieci	112
Obrazy bitmapowe	112
Grafika SVG	112
Element canvas	113
Element canvas — najlepsze rozwiązanie?	113
Rysowanie na elemencie canvas	114
Rysowanie ścieżek i figur	115
Łuki i okręgi	117
Zaawansowane style wypełniania i konturowania	121
Transformacje	124
Dołączanie tekstu, obrazu i cieni	126
Dodawanie tekstu	127
Zarządzanie stosem stanów	130
Rysowanie logo HTML5	131
Kompozycje	135
Manipulowanie danymi obrazu	136
Odczytywanie wartości pikseli	136
Uaktualnianie wartości pikseli	137
Eksport danych obrazu	139
Bezpieczeństwo i nałożone ograniczenia	139
Tworzenie efektów specjalnych na poziomie pikseli	140
Podsumowanie	141
Rozdział 7. Tworzenie szaty graficznej gry	143
Wstępne ładowanie plików	144
Dopasowanie wielkości obrazów	144
Modyfikowanie skryptu ładowania	145
Pasek postępu	149
Upiększanie tła	151
Konstruowanie ekranu gry	153
Nanoszenie planszy na płótno	154
Definiowanie planszy bez użycia płótna	160
Podsumowanie	165
Rozdział 8. Interakcja w grze	167
Identyfikowanie działań użytkownika	168
Zdarzenia obsługi myszy a ekrany dotykowe	168
Wirtualna klawiatura	168

Zdarzenia dotykowe	170
Zdarzenia sterowania i płótna	174
Konstruowanie modułu sterowania	176
Obsługa zdarzeń sterowania	177
Implementowanie mechaniki gry	180
Powiązanie kontrolki z funkcjami gry	184
Podsumowanie	189
Rozdział 9. Animowanie grafiki w grze	191
Wdrażanie interakcji	192
Synchronizacja animacji	193
Animowanie znacznika	196
Animowanie działań w grze	197
Punkty i czas	206
Tworzenie elementów interfejsu użytkownika	207
Tworzenie licznika czasu	210
Przyznawanie punktów	211
Koniec gry	219
Podsumowanie	222
Część III. Efekty 3D i dźwięk	223
Rozdział 10. Efekty dźwiękowe w grach	225
HTML5 i dźwięk	226
Wykrywanie obsługi audio	226
Wojny formatów — przebieg konfliktu	227
Odnajdywanie efektów dźwiękowych	229
Używanie elementu audio	229
Sterowanie odtwarzaniem	232
Dźwięk a urządzenia mobilne	235
Dane audio	235
Korzystanie z API danych Mozilla	236
Kilka przykładów	237
Konstruowanie modułu audio	241
Przygotowanie do odtwarzania dźwięku	242
Odtwarzanie efektów dźwiękowych	242
Zatrzymywanie dźwięku	244
Sprzątanie	244
Dołączanie efektów dźwiękowych do gry	245
Odtwarzanie dźwięku w ekranie gry	245
Podsumowanie	247
Rozdział 11. WebGL i grafika 3D	249
Trzeci wymiar w sieci	250
Wprowadzenie do WebGL	250
Debugowanie w WebGL	251
Tworzenie modułu pomocniczego	252
Shadery	252
Zmienne i typy danych	252
Shadery WebGL — praktyczne zastosowanie	256

Zmienne jednolite	260
Zmienne różnorodne	262
Renderowanie trójwymiarowych obiektów	262
Korzystanie z buforów wierzchołków	263
Używanie buforów indeksów	264
Korzystanie z modeli, widoków i projekcji	265
Macierz widoku modelu	266
Renderowanie	268
Ładowanie modeli Collada	271
Wykorzystywanie tekstur i oświetlenia	273
Dodawanie oświetlenia	273
Oświetlenie pikselowe	276
Tworzenie tekstur	278
Tworzenie modułu wyświetlania WebGL	283
Ładowanie plików WebGL	284
Przygotowanie modułu WebGL	285
Renderowanie klejnotów	287
Animowanie klejnotów	292
Podsumowanie	294

Część IV. Magazyn lokalny i tryb gry dla wielu graczy 295

Rozdział 12. Magazyn lokalny i tryb cache'owania 297

Magazynowanie danych — magazyn sieciowy	298
Używanie interfejsu magazynu	298
Konstruowanie modułu magazynu	301
Zapisywanie stanu gry na stałe	303
Zamknięcie gry	303
Pauzowanie gry	305
Zapisywanie danych gry	306
Tworzenie listy najlepszych wyników	308
Konstruowanie ekranu wyników	309
Przechowywanie rekordowych wyników	311
Wyświetlanie rekordowych wyników	312
Pamięć podręczna aplikacji	313
Manifest pamięci podręcznej	313
Podsumowanie	316

Rozdział 13. Technologia WebSocket i tryb wieloosobowy gry 317

Korzystanie z technologii WebSocket	318
Nawiązywanie połączenia z serwerem	318
Komunikacja w standardzie WebSocket	321
Stosowanie systemu Node.js po stronie serwera	322
Node.js — instalacja	323
Konstruowanie serwera HTTP	324
Konstruowanie pokoju czatu w technologii WebSocket	326
Podsumowanie	331

Rozdział 14. Dodatkowe zasoby	333
Korzystanie z oprogramowania pośredniczącego	334
Box2D	334
Impact	335
Three.js	337
Przystosowywanie gier do działania na urządzeniach mobilnych	338
PhoneGap	338
Appcelerator Titanium	341
Dystrybucja gier	342
Chrome Web Store	342
Zeewe	343
Google Play	344
App Store	344
Korzystanie z usług sieciowych	345
TapJS	345
Playtomic	345
JoyentCloud Node	346
Podsumowanie	346
Część V. Dodatki	349
Dodatek A. Element canvas — zbiór odwołań	351
Element canvas	352
API kontekstu dwuwymiarowego	352
Zarządzanie stanem	352
Transformacje	353
Figury i ścieżki	354
Wypełnienia i linie	356
Cienie	357
Obrazy	358
Tekst	358
Kompozycje	359
Manipulowanie pikselami	360
Dostępność	361
Dodatek B. WebGL — zbiór odwołań	363
API WebGL — zbiór odwołań	364
Typy danych	364
Typy tablic	365
Bufory	365
Shadery	366
Obiekty programów	367
Zmienne jednolite	368
Atrybuty wierzchołków	369
Rysowanie	370
Tekstury	371
Mieszanie i wtapianie	373
Bufor szablonu	374
Bufor głębi	375

Bufory renderowania	376
Bufory ramki	377
Inne funkcje	378
Parametry	380
Dodatek C. OpenGL Shading Language	385
Język GLSL ES — zbiór odwołań	386
Typy danych	386
Funkcje wbudowane	387
Wbudowane zmienne i stałe	393
Skorowidz	395

ROZDZIAŁ 4

Konstruowanie gry

W tym rozdziale:

- ▶ Tworzenie modułu planszy
- ▶ Zapisywanie stanu gry
- ▶ Mechanizmy układania klejnotów
- ▶ Wdrażanie reguł
- ▶ Reagowanie na zmianę położenia klejnotów

W tym rozdziale pokażę, w jaki sposób można zaimplementować reguły i mechanikę, które będą sterować działaniem gry. Przedstawię Ci kod niezbędny do stworzenia planszy, z którą pozostałe elementy programu będą mogły prowadzić interakcję. Zaprezentuję też metodę zachowania stanu gry poprzez odrębny moduł, który pozwoli na modyfikację planszy w pewnych ściśle określonych warunkach. Dodatkowo poprowadzę Cię przez konstrukcję kodu, który w odpowiedzi na dopasowania będzie odpowiednio zmieniał położenie pozostałych klejnotów.

Przedstawię również kilka problemów wynikających z zastosowania przez użytkownika różnych urządzeń sterujących rozgrywką. Gra dla jednej osoby działa w oparciu o lokalny kod uruchamiany po stronie klienta, który opisano w tym rozdziale. Zadbamy jednak także o serwerową implementację tych samych reguł.

Tworzenie modułu planszy

Kluczowa mechanika gry jest oddzielona od elementów wyświetlania i sterowania aplikacją. Moduł planszy, który będę omawiał w tym rozdziale, jest modelem danych stanu gry — a dokładniej: obecnego układu klejnotów. Moduł ten odsłania metody, które mogą zostać użyte przez inne moduły (głównie moduł ekranu gry) do interakcji ze stanem rozgrywki. Plansza ma służyć jako zaplecze ekranu gry, dlatego kod w tym rozdziale nie będzie zawierał żadnych nowych elementów wizualnych.

Jak sama nazwa wskazuje, moduł planszy jest logiczną reprezentacją samej planszy z klejnotami. Udostępnia ona innym modułom tylko dwie funkcje — funkcję zapytań, która służy do uzyskania dostępu do klejnotów, i funkcję umożliwiającą przemieszczanie brylantów. Funkcja przemieszczania zamienia miejscami tylko parę klejnotów, gdyż według zasad klejnoty mogą być przesuwane tylko zgodnie z ustalonymi regułami gry. Przemieszczenie klejnotów ma swoje konsekwencje: w jego wyniku klejnoty mogą zostać usunięte, co spowoduje pojawienie się nowych. To właśnie moduł planszy będzie odpowiedzialny za automatyczne spełnianie wszelkich warunków i różne zachowania. Został on zapisany w pliku *board.js* w folderze *scripts*. Listing 4.1 przedstawia jego pierwszą wersję.

Listing 4.1. Moduł planszy

```
jewel.board = (function() {
    /* funkcje gry */
    return {
        /* odsłonięte metody */
    };
})();
```

Odwołaj się teraz do pliku *board.js* w skrypcie *loader.js*, tak jak pokazano na listingu 4.2.

Listing 4.2. Ładowanie modułu planszy

```
// Ładowanie — etap 2.
if (Modernizr.standalone) {
    Modernizr.load([
        {
            load : [
                "scripts/screen.main-menu.js",
                "scripts/board.js"
            ]
        }
    ]);
}
```

Na tym etapie podstawowy moduł planszy nie daje nam żadnych opcji. Dlatego też nadszedł czas, byśmy go usprawnili i zajęli się jego pierwszą metodą.

Inicjalizowanie stanu gry

Kod planszy definiuje kilka ustawień, w tym: liczbę wierszy i kolumn, typy klejnotów itd. Tego rodzaju opcje najlepiej jest odseparować od samego kodu gry, tak by można było je łatwo zmienić bez potrzeby analizowania całego skryptu linijka po linijce. Listing 4.3 przedstawia nowy obiekt — `settings` — dodany do przestrzeni nazw w skrypcie `loader.js`.

Listing 4.3. Dodawanie obiektu settings

```
var jewel = {
  screens : {},
  settings : {
    rows : 8,
    cols : 8,
    baseScore : 100,
    numJewelTypes : 7
  }
};
```

Parametry `rows` i `cols` definiują rozmiar — odpowiednio — wierszy i kolumn. Plansza w *Brylantowym wojowniku* powstała na planie siatki 8×8, która bezproblemowo mieści się na małym ekranie. Z kolei ustawienie `baseScore` określa liczbę punktów, jaką gracz otrzymuje po usunięciu jednego klejnotu z łańcucha. Wynik ten jest mnożony w przypadku łańcuchów, w których znajdują się więcej niż trzy klejnoty. Ja ustawiłem podstawową liczbę punktów na 100, ale nie ma przeciwwskazań, byś ją zmienił, jeśli zależy Ci na uzyskaniu wyższych lub niższych wyników. Ostatnie nowe ustawienie — `numJewelTypes` — wskazuje liczbę typów klejnotów. Wartość ta odpowiada również liczbie sprite'ów (dwuwymiarowych obrazków) przedstawiających kamienie.

Nowe parametry są udostępniane pozostałej części gry i, co ważniejsze (przynajmniej na razie), modułowi planszy.

Inicjalizowanie planszy

Usprawnianie modułu zaczniemy od opracowania funkcji ustawiającej i inicjującej planszę. Nim właściwa gra się rozpocznie, plansza jest wypełniana losowymi klejnotami. Listing 4.4 prezentuje zawartość funkcji `initialize()` w skrypcie `board.js`.

Listing 4.4. Funkcja inicjalizująca

```
jewel.board = (function() {
  var settings,
      jewels,
      cols,
      rows,
      baseScore,
      numJewelTypes;

  function initialize() {
    settings = jewel.settings;
    numJewelTypes = settings.numJewelTypes,
    baseScore = settings.baseScore,
    cols = settings.cols;
    rows = settings.rows;
    fillBoard();
  }

  function print() {
```

```

var str = "";
for (var y = 0; y < rows; y++) {
  for (var x = 0; x < cols; x++) {
    str += getJewel(x, y) + " ";
  }
  str += "\r\n";
}
console.log(str);
}

return {
  initialize : initialize,
  print : print
};
})();

```

W listingu 4.4 z pewnością Twoją uwagę przykuły deklaracje zmiennych. Pierwsza zmienna importuje moduł ustawień, którego zawartość będzie nam za chwilę potrzebna. Druga zmienna — `jewels` — jest tablicą tablic, czyli dwuwymiarową tablicą reprezentującą stan planszy. W tablicy każdy klejnot jest reprezentowany przez liczbę całkowitą, która wskazuje jego typ. Wykorzystanie struktury tablicy upraszcza dostęp do konkretnych kamieni, w naturalny sposób tworząc system współrzędnych. Przykładowo poniższy fragment kodu pobiera klejnot z kratki o współrzędnych $x=3$, $y=2$:

```
var type = jewels[3][2];
```

Listing prezentuje również kilka zmiennych, których wartości wywodzą się z modułu `settings`. Opisuje je w następnym rozdziale. Funkcja `print()`, którą stworzyłem z myślą o debugowaniu, zwraca dane z planszy do analizy w konsoli JavaScript. Moduł planszy zainicjalizujesz, wpisując następującą komendę w okno konsoli:

```
jewel.board.initialize()
```

Jeśli zaś będziesz chciał sprawdzić generowane dane, wystarczy, że wprowadzisz następującą instrukcję, która wyświetli je w konsoli:

```
jewel.board.print()
```

Korzystanie z funkcji asynchronicznych

Nim przejdę do objaśniania kolejnej funkcji, wprowadźmy wspólnie małą modyfikację w funkcji `initialize()` — tak na przyszłość. W rozdziale 5. wyłożę Ci, jak użyć pracowników (ang. *web workers*), aby przemieścić wykonywanie kodu do oddzielnego wątku poprzez skonstruowanie modułu udostępniającego te same metody co stworzone w tym rozdziale. Pracownicy komunikują się z resztą aplikacji poprzez asynchroniczny interfejs programowania, a to oznacza, że również wszystkie metody publiczne odkryte przez moduł planszy muszą działać asynchronicznie.

Podobnie gdybyś dodał moduł planszy, który korzystałby z kodu po stronie serwera, odpowiedzialnego za weryfikację i walidację danych, musiałbyś wysłać asynchroniczne wywołania na serwer przy użyciu technologii Ajax. Każda funkcja, która modyfikuje stan gry, wymagałaby nawiązania dwukierunkowej komunikacji z serwerem, z tym że odpowiedź nie musiałaby być równoznaczna z otrzymaniem wyniku. Innymi słowy, sam fakt, że funkcja otrzymuje wywołanie zwrotne, nie oznacza, że wynik operacji jest gotowy.

Problem opóźnionej odpowiedzi można rozwiązać na kilka sposobów, w tym używając samodzielnego opracowanego systemu przydzielania zdarzeń lub mechanizmu **obiecanych obiektów** (ang. *promise objects*) znanego z bibliotek CommonJS i Node.js. Jednak najprostszym rozwiązaniem jest zapewnienie wywołania zwrotnego w formie argumentu dla właściwej metody, która wywoła funkcję

zwrotną po zakończeniu operacji. Prawdopodobnie znasz już taki wzorzec deklarowania z często używanych funkcji JavaScriptu w rodzaju `window.setTimeout()` lub `addEventListener()` dla elementów DOM. Funkcje te jako parametr przyjmują inne funkcje, które są wywoływane w pewnym momencie w trakcie wykonywania zadania. Listing 4.5 przedstawia zmodyfikowaną funkcję `initialize()` w skrypcie `board.js`.

Listing 4.5. Inicjalizowanie funkcji zwrotnej

```
jewel.board = (function() {
    ...
    function initialize(callback) {
        numJewelTypes = settings.numJewelTypes;
        baseScore = settings.baseScore;
        cols = settings.cols;
        rows = settings.rows;
        fillBoard();
        callback();
    }
    ...
})();
```

Aby zainicjalizować planszę poprzez konsolę JavaScript, użyj komendy:

```
jewel.board.initialize(function({})
```

Od tej pory wszystkie działania w funkcji `initialize()` będą wykonywane natychmiastowo, tak że rezultat będzie ten sam bez wywoływania funkcji zwrotnej. Różnica polega na tym, że wszelkie dokonane zmiany ułatwią późniejszą integrację modułu pracownika.

Wypełnianie planszy

Funkcja `fillBoard()` z listingu 4.5 generuje siatkę pól według wartości zmiennych `cols` i `rows` i wypełnia ją klejnotami. Listing 4.6 pokazuje jej zawartość po dodaniu do skryptu `board.js`.

Listing 4.6. Wypełnianie planszy klejnotami

```
jewel.board = (function() {
    ...
    function fillBoard() {
        var x, y;
        jewels = [];
        for (x = 0; x < cols; x++) {
            jewels[x] = [];
            for (y = 0; y < rows; y++) {
                jewels[x][y] = randomJewel();
            }
        }
    }
    ...
})();
```

Typ klejnotu jest wybierany przy użyciu funkcji `randomJewel()`, która po prostu zwraca liczbę całkowitą mieszczącą się pomiędzy 0 a $(\text{numJewelTypes} - 1)$. Listing 4.7 przedstawia funkcję `randomJewel()`.

Listing 4.7. Tworzenie losowego klejnotu

```

jewel.board = (function() {
  ...
  function randomJewel() {
    return Math.floor(Math.random() * numJewelTypes);
  }
  ...
})();

```

Podstawowy algorytm wypełniania planszy jest już gotowy. Przygotowane przez nas rozwiązanie jest jednak zawodne i nie gwarantuje przydatności otrzymanej planszy. Kamienie są wybierane losowo, więc istnieje szansa, że na planszy pojawią się jeden lub dwa gotowe łańcuchy. W fazie początkowej gra nie powinna ich jednak zawierać, ponieważ gracz mógłby zostać nagrodzony punktami za nic. Aby zagwarantować, że taka sytuacja nigdy nie będzie miała miejsca, funkcja `fillBoard()` musi układać klejnoty tak, żeby nie formowały się w rzędy więcej niż dwóch takich samych kamieni.

Algorytm wypełniania ustawia brylanty, zaczynając od górnego lewego rogu i kończąc na dolnym prawym. Oznacza to, że w trakcie wypełniania względem dowolnego kamienia inne klejnoty pojawiają się u góry i po jego lewej stronie. Łańcuch składa się z trzech identycznych kamieni, a to sprawia, że ułożony właśnie brylant nie może mieć tej samej barwy co dwa u góry lub dwa po jego lewej stronie. Dla niewielkiej planszy, takiej jak w *Brylantowym wojowniku*, takie proste rozwiązanie wystarczy. Listing 4.8 ilustruje zmiany w funkcji `fillBoard()`.

Listing 4.8. Usuwanie początkowych łańcuchów

```

jewel.board = (function() {
  ...
  function fillBoard() {
    var x, y,
        type;
    jewels = [];
    for (x = 0; x < cols; x++) {
      jewels[x] = [];
      for (y = 0; y < rows; y++) {
        type = randomJewel();
        while ((type === getJewel(x-1, y) &&
              type === getJewel(x-2, y)) ||
              (type === getJewel(x, y-1) &&
              type === getJewel(x, y-2))) {
          type = randomJewel();
        }
        jewels[x][y] = type;
      }
    }
  }
  ...
})();

```

Algorytm zawiera kilka pętli, które wybierają typ klejnotu, tak żeby nie powstał żaden łańcuch. W większości przypadków losowo wybrany klejnot nie stworzy łańcucha, ale gdyby jednak tak się stało, zostaje on zastąpiony innym.

Bez mechanizmu sprawdzania granicy pętli ta z listingu 4.8 próbowałaby uzyskać dostęp do kamieni spoza planszy, co spowodowałoby błąd. Dlatego też zamiast generować brylanty bezpośrednio, procedura `fillBoard()` korzysta z funkcji pomocniczej `getJewel()`, która zapobiega tego rodzaju uchybieniom. Listing 4.9 prezentuje zawartość tej funkcji.

Listing 4.9. Odczytywanie typu kamienia poprzez podanie jego współrzędnych

```

jewel.board = (function() {
  ...
  function getJewel(x, y) {
    if (x < 0 || x > cols-1 || y < 0 || y > rows-1) {
      return -1;
    } else {
      return jewels[x][y];
    }
  }
  ...
})();

```

Funkcja `getJewel()` zwraca cyfrę `-1`, jeśli którakolwiek ze współrzędnych wykracza poza granice planszy, tj. gdy wartość współrzędnej na osi x lub y jest mniejsza od zera lub większa od, odpowiednio, $(rows-1)$ i $(cols-1)$. Wszystkie prawidłowe typy kamieni mieszczą się w zakresie $[0; numTypes-1]$, a to gwarantuje, że zwracana wartość nigdy nie będzie wskazywać typu klejnotu, który uformowałby łańcuch.

Implementacja zasad

Teraz, gdy wstępna plansza jest gotowa, możemy przejść do mechaniki przemieszczania klejnotów. Nasz moduł zawiera metodę `swap`, która przyjmuje dwa zestawy współrzędnych jako parametry — po to, by zamienić miejscami klejnoty, na które one wskazują. Wszystkie zamiany miejsc, które nie spełniają reguł gry, są uznawane za nieważne. Zaczniemy od wdrażania mechanizmu walidacji ruchów gracza.

Walidacja ruchów gracza

Zamiana klejnotów miejscami jest ważna jedynie, jeśli jeden z nich uformuje łańcuch złożony z trzech lub więcej identycznych kamieni. W celu sprawdzenia poprawności zamian przygotowałem funkcję `checkChain()`, która analizuje, czy w nowym miejscu klejnot stanie się częścią łańcucha. Cała procedura sprawdzania polega na określeniu typu klejnotu na wskazanej pozycji, po czym przyrównaniu go do innych kamieni po lewej i po prawej, wraz z policzeniem, ile kamieni tego samego rodzaju znajduje się w bezpośrednim sąsiedztwie. Podobne przeszukiwanie odbywa się dla kierunków w górę i w dół. Jeśli suma wszystkich klejnotów w pionie lub poziomie będzie większa niż 2 (lub 3, jeśli w grę wchodzi środkowy klejnot łańcucha), dopasowanie zostanie uznane za ważne. Listing 4.10 przedstawia dokładną treść funkcji `checkChain()` w pliku `board.js`.

Listing 4.10. Sprawdzanie łańcuchów

```

jewel.board = (function() {
  ...
  // Zwraca liczbę w najdłuższym łańcuchu,
  // który zawiera kamień o współrzędnych (x,y).
  function checkChain(x, y) {
    var type = getJewel(x, y),
        left = 0, right = 0,
        down = 0, up = 0;

    // Sprawdza kamienie po prawej.
    while (type === getJewel(x + right + 1, y)) {
      right++;
    }
  }

```

```

    // Sprawdza kamienie po lewej.
    while (type === getJewel(x - left - 1, y)) {
        left++;
    }

    // Sprawdza kamienie u góry.
    while (type === getJewel(x, y + up + 1)) {
        up++;
    }

    // Sprawdza kamienie u dołu.
    while (type === getJewel(x, y - down - 1)) {
        down++;
    }
    return Math.max(left + 1 + right, up + 1 + down);
}
...
})();

```

Zwróć uwagę na to, że funkcja `checkChain()` nie zwraca wartości typu `boolean`, lecz liczbę klejnotów znalezionych w najdłuższym łańcuchu. Te wyniki dają nam trochę więcej danych na temat klejnotów, które można będzie wykorzystać podczas punktowania ruchu. Teraz, gdy już wiemy, jak się wykrywa łańcuchy, określenie, czy ruch jest ważny, będzie względnie łatwe. Pierwszym warunkiem jest to, że oba brylanty muszą znajdować się obok siebie. Tylko sąsiadujące klejnoty mogą zostać zamienione miejscami. Jeśli tak jest, gra powinna pozwolić na dokonanie tymczasowej zamiany. Zgodnie z wcześniejszymi ustaleniami, jeżeli ruch zostanie uznany za ważny, funkcja `checkChain()` zwróci liczbę większą niż 2 dla jednego lub dwóch wymiarów. Wystarczy więc przesunąć kamienie i zwrócić wartość wywołania `checkChain()`. Listing 4.11 pokazuje funkcję `canSwap()` zawartą w skrypcie `board.js`, która wdraża ten mechanizm walidacji.

Listing 4.11. Walidacja ruchu

```

jewel.board = (function() {
    ...
    // Zwraca wartość true, jeśli klejnot (x1,y1) może zostać zamieniony miejscem
    // z (x2,y2), tworząc dopasowanie.
    function canSwap(x1, y1, x2, y2) {
        var type1 = getJewel(x1,y1),
            type2 = getJewel(x2,y2),
                chain;

        if (!isAdjacent(x1, y1, x2, y2)) {
            return false;
        }
        // Tymczasowo zamienia miejscami wybrane kamienie.
        jewels[x1][y1] = type2;
        jewels[x2][y2] = type1;

        chain = (checkChain(x2, y2) > 2
            || checkChain(x1, y1) > 2);

        // Ustawia klejnoty na poprzednie miejsce.
        jewels[x1][y1] = type1;
        jewels[x2][y2] = type2;

        return chain;
    }
}

```

```

    }
    return {
        canSwap : canSwap,
        ...
    }
  })();

```

W listingu 4.8 wprowadzono nową funkcję pomocniczą: `isAdjacent()`. Funkcja zwraca wartość `true`, jeśli dwa podane zestawy współrzędnych wskazują na sąsiednie komórki, i `false`, jeżeli tak nie jest. Ich ustawienie względem siebie jest wyliczane na podstawie tzw. dystansu manhattańskiego. Jeśli dwie komórki przylegają do siebie, suma odległości musi wynosić 1. W listingu 4.12 znajdziesz treść funkcji `isAdjacent()`.

Listing 4.12. Sprawdzanie sąsiedztwa

```

jewel.board = (function() {
    ...
    // Zwraca wartość true, jeśli klejnot (x1,y1) sąsiaduje z kamieniem (x2,y2).
    function isAdjacent(x1, y1, x2, y2) {
        var dx = Math.abs(x1 - x2),
            dy = Math.abs(y1 - y2);
        return (dx + dy === 1);
    }
    ...
}

```

Działanie funkcji `canSwap()` przetestujesz w konsoli JavaScript po zainicjalizowaniu modułu planszy. Wykorzystaj też funkcję `print()` do sprawdzenia danych z planszy i upewnienia się, że kod działa prawidłowo, poprzez wpisywanie w konsoli instrukcji typu `jewel.board.canSwap(4,3,4,2)`.

Wykrywanie łańcuchów

Po dokonaniu zamiany gra przeszuka planszę w poszukiwaniu łańcucha i go usunie. Po wykonaniu ruchu zniknie zaledwie kilka klejnotów. Pojedynczy łańcuch powstaje w wyniku zamiany miejscami dwóch klejnotów. Jeżeli w wyniku ruchu jakieś klejnoty zostaną usunięte, w ich miejsce wejdą kamienie ponad nimi, a u góry planszy pojawią się nowe brylanty. Oznacza to, że należy sprawdzić obecność łańcuchów na planszy raz jeszcze — teraz jednak zadanie to nie będzie wcale takie proste. Jedyną opcją jest przeszukanie całej tablicy. Jeśli wykorzystasz do tego funkcję `checkChain()`, okaże się, że nie jest to takie skomplikowane. Listing 4.13 przedstawia funkcję `getChains()`, która przechodzi w pętli przez wszystkie kamienie w poszukiwaniu łańcuchów.

Listing 4.13. Przeszukiwanie planszy

```

jewel.board = (function() {
    ...
    // Zwraca dwuwymiarową mapę długości łańcuchów.
    function getChains() {
        var x, y,
            chains = [];
        for (x = 0; x < cols; x++) {
            chains[x] = [];
            for (y = 0; y < rows; y++) {
                chains[x][y] = checkChain(x, y);
            }
        }
    }
}

```

```

        return chains;
    }
    ...
})();

```

Zwrócona przez funkcję `getChains()` zmienna `chains` jest dwuwymiarową mapą planszy. Zamiast danych o typach kamieni mapa ta zawiera informacje o łańcuchach, na które składają się klejnoty. Każda komórka na planszy jest sprawdzana poprzez wywołanie metody `checkChain()`, co skutkuje dopasowaniem odpowiedniej komórki na mapie do zwracanej przez funkcję wartości.

Usuwanie łańcuchów klejnotów

Samo odnalezienie łańcucha nie wystarczy. Gra musi również wykorzystać otrzymaną informację. Dokładniej rzecz ujmując, łańcuchy muszą zostać usunięte, a klejnoty ponad nimi powinny opaść na ich miejsce. Mapa łańcuchów jest przetwarzana w funkcji `check()` widocznej w listingu 4.14.

Listing 4.14. Przetwarzanie łańcuchów

```

jewel.board = (function() {
    ...
    function check() {
        var chains = getChains(),
            hadChains = false, score = 0,
                removed = [], moved = [], gaps = [];

        for (var x = 0; x < cols; x++) {
            gaps[x] = 0;
            for (var y = rows-1; y >= 0; y--) {
                if (chains[x][y] > 2) {
                    hadChains = true;
                    gaps[x]++;
                    removed.push({
                        x : x, y : y,
                        type : getJewel(x, y)
                    });
                } else if (gaps[x] > 0) {
                    moved.push({
                        toX : x, toY : y + gaps[x],
                        fromX : x, fromY : y,
                        type : getJewel(x, y)
                    });
                    jewels[x][y + gaps[x]] = getJewel(x, y);
                }
            }
        }
    }
    ...
})();

```

Funkcja ta usuwa klejnoty z planszy i przesuwa w ich miejsce nowe. Poza zmodyfikowaniem planszy funkcja `check()` zbiera informacje o usuniętych i przesuniętych kamieniach w dwóch tablicach — `removed` i `moved`. Dane te są ważne, gdyż przydają się później np. przy animowaniu zmian na ekranie.

Wykorzystując zagnieżdżone pętle, funkcja `check()` przechodzi przez wszystkie komórki na planszy. Jeśli dana komórka została oznaczona na mapie wartością większą niż 2, informacja o umiejscowieniu i typie klejnotu zostaje zapisana w tablicy `removed` z wykorzystaniem literału obiektu. Kamień,

który opadnie w nowe miejsce, nadpisze dane o pozycji w późniejszym etapie, toteż na razie nie trzeba modyfikować tablicy klejnotów.

Zwróć uwagę na sposób, w jaki pętla sprawdza wiersze: od dołu do góry zamiast z góry na dół. Rozwiązanie to pozwoli Ci natychmiastowo przemieścić inne klejnoty w dół planszy. Algorytm ten zachowuje licznik każdej kolumny wewnątrz tablicy gaps. Nim zacznie on przetwarzać kolejną kolumnę, ustawia jej licznik na 0. Za każdym razem gdy usunięty zostanie klejnot, licznik jest zwiększany o 1. Z kolei jeżeli klejnot pozostaje na swoim miejscu, licznik tablicy gaps określi, czy powinien on zostać przesunięty w dół. Stanie się tak, jeżeli licznik ma wartość dodatnią — wtedy klejnot opadnie w dół o równą mu liczbę wierszy. Wartość ta jest zapisana w drugiej tablicy — moved — za pomocą podobnego literału obiektu, z tym że tym razem zachowane zostaną w niej pozycja początkowa i końcowa. W tym momencie należy uaktualnić tablicę jewels, ponieważ wskazywane przez nią współrzędne uległy zmianie.

Generowanie nowych klejnotów

Funkcja check() nie została do końca opracowana; wciąż zawiera kilka niedociągnięć. Przenosząc istniejące klejnoty w dół, wypełniasz co prawda luki, lecz tworzysz nowe w górnej części planszy. Dlatego też po przetworzeniu wszystkich klejnotów w kolumnie należy stworzyć nowe kamienie, które spłyną z górnej granicy w dół. Listing 4.15 przedstawia modyfikację odpowiedzialną za ten mechanizm.

Listing 4.15. Dodawanie nowych klejnotów

```

jewel.board = (function() {
  ...
  function check() {
    ...
    for (var x = 0; x < cols; x++) {
      gaps[x] = 0;
      for (var y = rows-1; y >= 0; y--) {
        ...
      }

      // Dodaje nowe klejnoty u góry planszy.
      for (y = 0; y < gaps[x]; y++) {
        jewels[x][y] = randomJewel();
        moved.push({
          toX : x, toY : y,
          fromX : x, fromY : y - gaps[x],
          type : jewels[x][y]
        });
      }
    }
  }
  ...
})();

```

Liczba nowych kamieni, które trzeba wygenerować w kolumnie, jest równa liczbie wolnych komórek, które się w niej znajdują. Konkretnie współrzędne, które mają zająć klejnoty, są łatwe do obliczenia, ponieważ nowe kamienie zawsze spadają na wolne miejsca z góry planszy. Informacje o nowych klejnotach są dodawane do tablicy moved poza istniejącymi wcześniej brylantami, które przeniesiono niżej. Z uwagi na fakt, że nowe klejnoty nie mają współrzędnych początkowych, wprowadziłem nieistniejące współrzędne spoza planszy, tak jakby nowe klejnoty istniały wcześniej i czekały na swoją kolej.

Przyznawanie punktów

W funkcji `initialize()` wprowadziłem zmienną `baseScore`, na podstawie której będę obliczał liczbę przyznawanych punktów. Listing 4.16 prezentuje kod odpowiedzialny za punktowanie ruchów gracza, dodany w skrypcie `check()`.

Listing 4.16. Przyznawanie punktów za uformowane łańcuchy

```

jewel.board = (function() {
  ...
  function check() {
    ...
    for (var x = 0; x < cols; x++) {
      gaps[x] = 0;
      for (var y = rows-1; y >= 0; y--) {
        if (chains[x][y] > 2) {
          hadChains = true;
          gaps[x]++;
          removed.push({
            x : x, y : y,
            type : getJewel(x, y)
          });

          // Dodaje punkty do wyniku.
          score += baseScore
            * Math.pow(2, (chains[x][y] - 3));
        } else if (gaps[x] > 0) {
          ...
        }
      }
    }
  }
}());

```

Za każdy składowy klejnot łańcucha gra dodaje punkty do ogólnego wyniku. Liczba otrzymanych punktów zależy od długości łańcucha. Każdy dodatkowy łańcuch podwaja wynik.

Pamiętaj: Zmienna `score` nie zawiera całkowitego wyniku gracza; to jedynie suma punktów zebrana w rezultacie wykonania funkcji `check()`. Moduł planszy nie odzwierciedla reprezentacji idei gracza. Po prostu oblicza, jaką liczbę punktów należy nadać za wykonanie danego ruchu.

Łańcuchy zniknęły, a powstałe luki zostały wypełnione nowymi klejnotami. Trzeba jednak pamiętać, że nowe klejnoty mogą stworzyć nowe łańcuchy — dlatego przed nami jeszcze trochę pracy. Funkcja `check()` będzie wywoływać siebie rekurencyjnie do czasu, gdy nie wykryje żadnych łańcuchów. Jak pamiętamy, musi ona zwracać zachowane zmiany. Listing 4.17 przedstawia wprowadzone w niej zmiany.

Listing 4.17. Rekurencyjna analiza planszy

```

jewel.board = (function() {
  ...
  function check(events) {
    ...

```

```

events = events || [];
if (hadChains) {
  events.push({
    type : "remove",
    data : removed
  }, {
    type : "score",
    data : score
  }, {
    type : "move",
    data : moved
  });
  return check(events);
} else {
  return events;
}
...
}

```

Teraz należy połączyć dane z tablic removed, moved i score z informacjami zwracanymi przez rekurencyjne wywołania. W tym celu dodaj opcjonalny argument zdarzenia w funkcji check(). Argument ten jest używany tylko w rekurencyjnych wywołaniach. Jeśli do funkcji nie zostanie przekazany żaden argument, zdarzenia są przypisywane do pustej tablicy. Po wykonaniu analizy planszy dodajemy wynik, po czym plansza zmienia się w tablicę zdarzeń poprzez użycie prostego formatu obiektów zdarzeń, który przedstawiłem w listingu 4.16. Wszystkie obiekty zdarzeń zawierają właściwości type i data. Jeżeli gra nie znajdzie żadnych łańcuchów, ułożenie kamieni na planszy nie ulegnie zmianie, a funkcja check() nie zostanie wywołana ponownie. W tym momencie zostanie zwrócona tablica zdarzeń, która odbierze pierwsze wywołanie i wykona żadaną operację. W ten sposób podmiot, który ją wywoła, uzyska listę wszystkich zmian, które zaszły, nim gracz wykona kolejny ruch.

Ponowne wypełnianie siatki

W trakcie rozgrywki prędzej czy później dojdzie do sytuacji, w której ułożenie kamieni na planszy zablokuje wszystkie ruchy. Gra musi rozpoznać taką sytuację i zainicjować ponowne wypełnienie planszy nowymi klejnotami, by gracz mógł kontynuować zabawę. Przygotujemy więc funkcję, która rozpozna, czy na planszy możliwe są jakiegokolwiek ruchy. Listing 4.18 przedstawia treść funkcji hasMoves().

Listing 4.18. Sprawdzanie dostępnych ruchów

```

jewel.board = (function() {
  ...
  // Zwraca wartość true, jeśli znajdzie przynajmniej jeden możliwy ruch.
  function hasMoves() {
    for (var x = 0; x < cols; x++) {
      for (var y = 0; y < rows; y++) {
        if (canJewelMove(x, y)) {
          return true;
        }
      }
    }
  }
  return false;
}
...
})();

```

Funkcja `hasMoves()` zwraca wartość `true`, jeśli co najmniej jeden klejnot może zostać przeniesiony, tak by stworzył łańcuch; w przeciwnym razie zwraca wartość `false`. W listingu 4.19 znajdziesz pomocniczą metodę `canJewelMove()`, której zadaniem jest sprawdzenie współrzędnych dla ruchów.

Listing 4.19. Sprawdzanie wariantów ruchu dla pojedynczego kamienia

```
jewel.board = (function() {
    ...
    // Zwraca true, jeśli współrzędne (x,y) są odwzorowane na planszy i
    // jeśli kamień w tym punkcie może pozostać zamieniony z sąsiednim.
    canJewelMove(x, y).
    function canJewelMove(x, y) {
        return ((x > 0 && canSwap(x, y, x-1, y)) ||
            (x < cols-1 && canSwap(x, y, x+1, y)) ||
            (y > 0 && canSwap(x, y, x, y-1)) ||
            (y < rows-1 && canSwap(x, y, x, y+1)));
    }
    ...
})();
```

Aby sprawdzić, czy klejnot może zostać przeniesiony i stworzyć łańcuch, metoda `canJewelSwap()` używa funkcji `canSwap()` — to właśnie ona określa możliwość zamiany z jednym z czterech sąsiadujących kamieni. Do wywołania metody `canSwap()` dochodzi, jeśli sąsiadujący kamień znajduje się w obrębie planszy. W rezultacie funkcja ta dokonuje zamiany z kamieniem np. po lewej stronie, tylko jeśli jego współrzędna wynosi przynajmniej 1 i jest mniejsza od `(cols-1)`.

Jeżeli gra odkryje, iż nie da się wykonać żadnego ruchu (co nastąpi, gdy funkcja `hasMoves()` zwróci `false`), plansza zostanie automatycznie wypełniona nowymi brylantami. Wyzwalacz wypełniania umieściłem w funkcji `check()`. Po zidentyfikowaniu łańcuchów, usunięciu klejnotów i wygenerowaniu nowych kamieni wywołana jest funkcja `hasMoves()` i w przypadku niestwierdzenia możliwości ruchu wprowadza nowy zestaw brylantów na planszy. Wszelkie opisane zmiany znajdziesz w listingu 4.20.

Listing 4.20. Wyzwalanie wypełnienia planszy

```
jewel.board = (function() {
    ...
    function check(events) {
        ...
        if (hadChains) {
            ...
            // Wypełnia planszę ponownie, jeśli gracz nie będzie miał żadnego ruchu.
            if (!hasMoves()) {
                fillBoard();
                events.push({
                    type: "refill",
                    data: getBoard()
                });
            }
            return check(events);
        } else {
            return events;
        }
    }
})();
```

Poza wyzwoleniem metody `fillBoard()` w listingu pojawia się zdarzenie wypełniania dodane do tablicy zdarzeń. Zawiera ono kopię planszy stworzoną przez funkcję `getBoard()` z listingu 4.21.

Listing 4.21. Kopiowanie danych z planszy

```

jewel.board = (function() {
  ...

  // Tworzy kopię planszy z klejnotami.
  function getBoard() {
    var copy = [],
        x;
    for (x = 0; x < cols; x++) {
      copy[x] = jewels[x].slice(0);
    }
    return copy;
  }

  return {
    ...
    getBoard : getBoard
  };
})();

```

Proste wywołanie metody `fillBoard()` nie gwarantuje, że na nowej planszy pojawi się możliwość wykonania ruchu. Istnieje szansa, że losowe wybieranie klejnotów uniemożliwi jakiegokolwiek działanie gracza. Takie ułożenie planszy powinno wyzwolić kolejne, niewidoczne dla użytkownika wypełnienie. Najlepszym miejscem na umieszczenie tego mechanizmu jest funkcja `fillBoard()`. Pojedyncze wywołanie metody `hasMoves()` określi, czy plansza nadaje się do dalszej gry — jeśli okaże się, że nie, funkcja `fillBoard()` zacznie wywoływać siebie rekurencyjnie do czasu, aż uzyska pożądaną wynik. W rezultacie plansza będzie wypełniana raz po raz, do czasu gdy uzyskana zostanie przynajmniej jedna para klejnotów. Listing 4.22 prezentuje mechanizm dodany do funkcji `fillBoard()`.

Listing 4.22. Rekurencyjne wypełnianie planszy klejnotami

```

jewel.board = (function() {
  ...
  function fillBoard() {
    ...

    // Rekurencyjnie wypełnia planszę, jeśli nie ma na niej żadnych ruchów.
    if (!hasMoves()) {
      fillBoard();
    }
  }
  ...
})();

```

W ten sposób mechanizm wypełniania poradzi sobie z hipotetycznym scenariuszem, w którym na wstępnej planszy nie byłoby żadnych ruchów. Jednakże istnieje drobne prawdopodobieństwo, że pierwsza wczytana plansza nie pozwoli graczowi na wykonanie żadnego dopasowania. Wywołanie rekurencyjne rozwiązuje ten problem.

Dopasowywanie brylantów

Przygotowaliśmy wszystkie funkcje zarządzające stanem planszy. Brakuje nam tylko metody zamieniającej klejnoty miejscami. Jej opracowanie nie jest trudne. Dysponujesz już funkcją `canSwap()`, która określa, czy gracz może wykonać danych ruch, oraz metodą `check()`, która zajmuje się wydarzeniami po wykonaniu dopasowania. Listing 4.23 przedstawia treść funkcji `swap()`.

Listing 4.23. Funkcja `swap()`

```

jewel.board = (function() {
    ...
    // Jeśli istnieje taka możliwość, zamienia miejscami klejnot w komórce (x1,y1)
    // z klejnotem w komórce (x2,y2).
    function swap(x1, y1, x2, y2, callback) {
        var tmp,
            events;

        if (canSwap(x1, y1, x2, y2)) {

            // Zamienia klejnoty miejscami.
            tmp = getJewel(x1, y1);
            jewels[x1][y1] = getJewel(x2, y2);
            jewels[x2][y2] = tmp;

            // Sprawdza planszę i pobiera listę zdarzeń.
            events = check();

            callback(events);
        } else {
            callback(false);
        }
    }

    return {
        ...
        swap : swap
    };
})();

```

Funkcja `swap()` zostanie odsłonięta dla pozostałej części kodu i może wpłynąć na stan planszy, toteż musi ona działać zgodnie z tym samym asynchronicznym mechanizmem co funkcja `initialize()`. Dlatego też poza dwoma zestawami współrzędnych metoda `swap()` przyjmuje jako parametr funkcję zwrrotną. W zależności od tego, czy ruch użytkownika się powiedzie, wywołanie zwrótnie otrzyma jako parametr albo listę zdarzeń, albo wartość `false` (w przypadku nieuznanego ruchu). Listing 4.24 przedstawia funkcje upublicznione poprzez moduł planszy.

Listing 4.24. Zwrocenie metod publicznych

```

jewel.board = (function() {
    ...
    return {
        initialize : initialize,
        swap : swap,
        canSwap : canSwap,
        getBoard : getBoard,
        print : print
    };
})();

```

To wszystko. Dzięki odsłonięciu tych funkcji stan gry może zostać zmodyfikowany jedynie wskutek ustawienia nowej planszy lub wywołania metody `swap()`. Od tego momentu wszystkie zasady gry będą egzekwowane przez funkcję `swap()`, co jest gwarancją integralności planszy. Dodatkowo jedynym punktem wejściowym danych jest funkcja `getBoard()`, która nie pozwala nadpisywać informacji, co zmniejsza ryzyko „złamania” reguł przez resztę kodu.

Jak zwykle działanie metody `swap()` przetestujesz, wywołując ją w konsoli. Aby tego dokonać, wpisz:

```
jewel.board.initialize(function({})
```

Następnie użyj komendy `jewel.board.print()`, aby znaleźć właściwą współrzędną, i wpisz np.:

```
jewel.board.swap(4,3,4,2, function(e){console.log(e)})
```

Pamiętaj przy tym, że funkcja `swap()` wymaga wywołania zwrotnego. Przygotuj własnoręcznie metodę, która będzie zwracać listę zdarzeń w konsoli.

Podsumowanie

W rozdziale tym wyłożyłem Ci sposób wdrożenia elementarnej mechaniki gry. Przeprowadziłem Cię przez implementację wszystkich jej reguł — w tym dotyczących przemieszczania klejnotów, usuwania łańcuchów i generowania dodatkowych klejnotów. Plansza gry została zrealizowana w obrębie jednego modułu, pozwalając na dostęp do informacji w zaledwie kilku miejscach, co gwarantuje, że wprowadzone modyfikacje będą działały według wyłożonych wcześniej zasad.

Ponadto rozdział ten bierze pod uwagę tryb dla wielu graczy, którego wdrożeniem zajmiemy się w dalszej części książki. Na razie zapewniliśmy, że gra będzie korzystała z kodu lokalnego lub specjalnego skryptu udostępnionego przez serwer. Użycie funkcji zwrotnych w kilku kluczowych metodach pozwoli dwóm opracowanym modułom współdzielić ten sam interfejs, ułatwiając dodanie asynchronicznego, serwerowego modułu planszy w dalszej fazie projektowania gry.

Skorowidz

A

ActionScript, 334
ADB, *Patrz:* Android Debug Bridge
Adobe Edge, 25
Adobe Flash Professional, 340
adres
 IP, 346
 URL, 318, 330
AIR, 340
Ajax, 23, 80, 96, 98, 271, 318, 322
aLogcat, 73
Amazon Elastic Compute Cloud, 324
Amazon Web Services, 324
ambient light, *Patrz:* światło otaczające
Android, 25, 46, 47, 60, 61, 68, 69, 70, 73, 97, 163, 168, 170, 171, 235, 250, 334, 335, 338, 339, 341, 343, 344
Android Debug Bridge, 73
Android Developer Console, 344
Angry Birds, 334
animacja, 113, 188, 193, 197, 211, 219, 268, 292
 cykl, 196
 cykl wewnętrzny, 193
 zoomfade, 217
API, 21, 23, 37, 113
 audio, 236, 237, 239
 czasowe, 196
 danych Mozilla, 236
 FileSystem, *Patrz:* FileSystem API
 IndexedDB, 298
 JavaScriptu, 345
 komunikacyjne, 97, 98
 linii, 115
 magazynu, 299
 obiektu kontekstu
 dwuwymiarowego, 352
 standaryzowane, 318

synchronizacji animacji, 193, 194
ścieżki, 115, 117
WebGL, 282, 364
WebSocket, 318, 326
aplikacja, 62, 63, 68
 czatu, 326
 debugowanie, 71
 ikonka, 65
 natywna, 334, 338, 340
 śledzenia promieni, *Patrz:* śledzenie promieni
App Store, 338, 344
Appcelerator Titanium, 341, 344
Apple, 61, 163
Apple App Store, 47, 334
Application Programming Interface, *Patrz:* API
Aptana Studio, 341
Arcade Fire, 338
arkusz stylów, 34, 144, 145, 160, 163, 304
 mobilny, 59
Audacity, 229

B

Bada, 339
base64, *Patrz:* format base64
baza danych, 323
Bejeweled, 27, 30
Béziera
 krzywa, *Patrz:* krzywa Béziera
 płaszczyzna, *Patrz:* płaszczyzna Béziera
biblioteka
 glMatrix, 265, 266, 267
 jQuery, 33
 Modernizr, 26, 27, 33, 36, 63, 108, 152, 170, 217, 227, 228, 284
 Prototype, 33, 334
 Sizzle, 272
 Sound Manager 2, 227

SoundManager, 22
 yepnope.js, 27
Biolab Disaster, 335
BlackBerry, 341
BlackBerry WebWorks, 339
Blender, 271, 338
Box2D, 334, 337
Box2DJS, 334
bufor, 263, 321, 365
 barw, 268
 głębi, 268, 375
 indeksu, 264
 normalnych, 274
 ramki, 377
 renderowania, 376
 szablonu, 268, 374
 wierzchołków, 264, 265, 269, 274

C

Cabello Ricardo „Mr. Doob”, 337
cache, *Patrz:* pamięć podręczna
canvas, 20, 21, 25, 27, 28, 112, 113, 114, 139, 152, 154, 174, 237, 250, 251, 262, 334, 338, 352, 361, 370
 cieniowanie, 129
 czyszczenie, 268
 efekty, 126, 129
 elementy awaryjne, 129
 obracanie, 126
 ograniczenia, 139
 operacje na pikselach, 136, 137
 skalowanie, 114, 125
 tekst, 127, 128
 transformacja, 124, 125, 138
 transformacja trójwymiarowa, 204
 translacja, 125, 126
chmura, 324
Chrome, 37, 41, 97, 98, 100, 136, 193, 250, 298, 318, 342, 346

Chrome Web Store, 342

Chromium, 251

cieniowanie, 129, 357

Phonga, 273

clipping path, *Patrz*: ścieżka przycinająca

Collada, 271, 272, 274, 281

Comet, 23

cookie, 24, 298, 302, 318

Crayon Physics Deluxe, 334

CSS3, 20, 25, 28, 57, 59

Cut the Rope, 48

czat, 326, 329

czytnik ekranu, 361

D

dane

numeryczne, 253

typ, 253, 364, 386

wektorowe, 253

źródłowe obrazu, 27

Danger Mouse, 338

debugowanie, 37, 71, 251, 336

dedicated worker, *Patrz*:

pracownik sieciowy dedykowany

delegacja zdarzenia, 56

demoscena, 113

deseń, 27, 124, 156, 163, 357,

Patrz też: tekstura

diffuse light, *Patrz*: światło

rozproszenia

DirectX, 252

display, *Patrz*: moduł wyświetlania

div, *Patrz*: element div

Dragonfly, 71

Dreamweaver, 340

dźwięk, 22, 226, 242, 338

wizualizacja, 237

zapisywanie, 237

zatrzymywanie, 244

E

Eclipse, 341

ECMAScript, 148

edytor

poziomów, 336

tekstu, 300

efekty

dźwiękowe, 229, 235, 242

fizyczne, 334

kompozycji, 27

specjalne, 66, 126, 140, 196,

219, 290, 305, 373

ekran

dotykowy, 46, 168, 176, 177

gry, 153

instalacyjny, 64, 65

powitalny, 31, 39, 40, 41, 54,

55, 63, 145, 149

wielkość, 49, 59

eksplozja, 220

element

audio, 22, 23, 28, 226, 228,
229, 230, 232, 235, 339

bgsound, 226

canvas, *Patrz*: canvas

div, 21, 34, 151, 171, 215

DOM, 33, 97, 99, 160

img, 21

progress, 207

video, 226

em, 50, 54, 162

enkapsulacja, 38

experimental-webgl, 251

ExplorerCanvas, 27

Extensible Markup Language,

Patrz: XML

F

Fast Fourier Transform, 239

FileSystem API, 298

Firebug, 71

Firefox, 37, 41, 97, 98, 100, 136,

193, 194, 195, 231, 236, 250,

298, 318, 319, 320

Flash, 22, 25, 27, 226, 227, 229,

340

font

Geo Regular, 40

optymalizacja, 41

sieciowy, 28, 40, 42, 127

Slackey Regular, 40

TrueType, 40

wysokość, 50

FontSquirrel, 40

format

AAC, 227

audio, 227

base64, 21

BMP, 139

GIF, 112

JPEG, 112, 139

MP3, 227

OBJ, 338

Ogg Vorbis, 227

PNG, 112, 139

SVG, 112, 113

Wavefront OBJ, 338

WebM, 227

WOFF, 40

XML, 113

Freesound Project, 229

Fruit Ninja, 48

funkcja

linii, 115

matematyczna, 387

nienazwana, 38

trygonometryczna, 389

G

Generator tonów, 239

gesture, *Patrz*: zdarzenie gestów

GitHubset, 340

GLSL, *Patrz*: język GLSL

głośność, 233, 235

gniazdo, 23

Google, 27, 334

Google Analytics, 345

Google Checkout, 344

Google Play, 47, 338, 344

Google Web Fonts, 40

GPU, *Patrz*: procesor graficzny

gra

dystrybucja, 342

hosting, 334, 345

inicjowanie, 79

kontener, 35

mechanika, 78, 83, 180

moduł, 38

opcje zapisu, 24

pauzowanie, 305

plansza, 78, 79

przywrócenie danych, 306

reguły, 78

warstwa, 31

wersja demo, 345

wieloosobowa, 23

wieloplatformowa, 28

wynik, *Patrz*: wynik

zamknięcie, 303

gradient, 124, 357

CSS3, 162

kołowy, 123, 153

liniowy, 123

grafika, 112, 154

3D, 262

bitmapowa, 112, 114, 139

powitalna, 67

skalowanie, 112

SVG, *Patrz też*: format SVG

trójwymiarowa, 338

wektorowa, 20

Graphic Processing Unit, *Patrz*:

procesor graficzny

H

High-Level Shading Language,

Patrz: HLSL

HLSL, 252

HTTP request, *Patrz*: żądanie HTTP

I

identyfikator
 URI, 21
 IETF, 318
 IEWebGL, 250
 ikonka
 aplikacji, 65
 img, *Patrz:* element img
 immediate mode, *Patrz:* tryb natychmiastowy
 Impact, 335, 337, 345
 IndexedDB, *Patrz:* API IndexedDB
 instancja, 324
 interfejs, 342
 aplikacji sieciowej, 342
 asynchroniczny, 80, 104
 MediaElement, 226
 odtwarzania, 234
 programowania, 80
 programowania aplikacji, *Patrz:* API
 użytkownika, 113
 Internet Engineering Task Force, *Patrz:* IETF
 Internet Explorer, 24, 25, 27, 37, 41, 97, 129, 136, 194, 226, 250, 298, 318
 iOS, 25, 40, 46, 61, 62, 97, 168, 170, 171, 235, 318, 334, 335, 338, 339, 341
 iPad, 46, 59, 61, 68, 174, 335, 343
 iPhone, 67
 iPhone, 46, 61, 174, 335, 343
 iPod, 46, 61
 iPod touch, 67
 Irish Paul, 41

J

JavaScript, 21, 22, 25, 62, 96, 168, 193, 334, 364
 JavaScript Object Notation, *Patrz:* JSON
 jednostki
 em, *Patrz:* em
 statyczne, 50
 względne, 50
 język
 GLSL, 252, 255, 258, 386
 OpenGL Shading Language, 250
 PHP, 336
 Python, 338
 serwerowy, 322
 XML, 271
 Joyent, 324
 jQuery Mobile, 343
 JSON, 24, 99, 271, 321, 343

K

kanał alfa, 136, 138, 197
 kaskadowy arkusz stylów, *Patrz:* CSS3
 kerning, 50
 Khronos, 237, 250, 271
 klawiatura, 47, 168, 176, 177, 179
 wirtualna, 47, 168
 klient
 użytkownika, 25
 klucz, 299
 SSH, 346
 kodek dźwiękowy, 22
 kolor czyszczenia, 262
 kompatybilność, 40
 wsteczna, 20, 25
 kompatybilność, 302
 komponent
 semantyczny, 207
 span, 207
 kompozycja, 135, 136, 138, 359
 kontekst, 114, 119, 250
 trójwymiarowy, 250, 251
 krzywa Béziera, 119, 120

L

laptop, 49
 licznik
 czasu, 193, 198, 206, 210, 211, 305
 punktów, 206, 211
 linia, *Patrz też:* ścieżka, krzywa bazowa, 128
 grubość, 116
 prosta, 115
 Linux, 341
 lista, 148

Ł

łańcuch, 85, 86
 tekstowy, 24, 321
 łuk, 117

M

Mac OS, 341
 macierz, 255, 261, 265, 267, 390
 projekcji, 265, 267
 transformacji, 125, 126, 353
 translacji, 125
 widoku modelu, 265, 266, 274, 290
 magazyn
 lokalny, 24, 298, 301
 sesji, 24, 298, 301
 sieciowy, 24, 298, 299

manifest, 313, 314, 315, 343
 margines, 35
 maska, 120
 media query, *Patrz:* zapytanie medialne CSS3
 menu
 główne, 31, 52, 55, 145
 kontekstowe, 48
 metoda
 asynchroniczna, 80, 322
 audio.canPlayType, 228
 audio.mozCurrentSampleOffset, 241
 audio.mozSetup, 237
 audio.stop, 244
 canPlayType, 226
 ctx.createImageData, 360
 ctx.getImageData, 360
 deseni, 356
 document.createElement, 226
 gl.getError, 251
 gl.getParameter, 380
 gradientów, 356
 malowania obrazów, 358
 Math.sin, 197
 operowania na pikselach, 360
 play, 232
 prostokątów, 355
 publiczna, 80
 requestAnimationFrame, 193, 194, 195, 211
 require, 324
 response.end, 325
 response.write, 325
 response.writeHead, 325
 setInterval, 193, 194
 setTimeout, 211
 setTimout, 193, 194, 195
 statyczna, 266
 ścieżek, 354
 teksty, 358
 transformacji, 353
 ws.close, 319
 ws.send, 321
 zarządzająca stosem stanów, 352
 middleware, *Patrz:*
 oprogramowanie pośredniczące
 mieszanie barw, 288
 mipmapa, 279
 model
 DOM, 160, 322
 Phongą, 273, 275
 Modernizr, *Patrz:* biblioteka Modernizr
 Module Pattern, *Patrz:* wzorzec modułów

moduł

- audio, 226, 241
 - HTTP, 324
 - magazynu, 301, 303
 - pauzy, 305
 - sterowania, 176, 185
 - wyników, 308
 - wyświetlania, 154, 158, 160, 196, 252, 283, 284
 - wyświetlania WebGL, 284
 - zewnątrzny, 323
- multi-touch event, *Patrz:*
zdarzenie wielodotykowe
- mysz, 48, 168, 170, 175, 176, 177

N

- natura bezstanowa, 24
 - nie-tablica, 148
 - Nitobi, 340
 - Nobi, 338
 - Node Package Manager, 323
 - Node SmartMachines, 324
 - Node.js, 23, 33, 318, 322, 324, 326, 346
 - instalacja, 323
- notacja
JSON, *Patrz:* JSON

O

obiekt

- arguments, 148
 - buforów, 263, 365
 - dotyku, 179
 - kontekstu dwuwymiarowego, 352
 - kontekstu graficznego, 114, 126, 129
 - localStorage, 298, 301
 - magazynu, 300
 - obiecany, 80
 - port, 101
 - pracownika sieciowego, 103
 - programu, 259, 367
 - resource, 146
 - sessionStorage, 301
 - shadera, 259
 - tabel, 365
 - trójwymiarowego kontekstu, 251
 - trójwymiarowy, 262
 - WebSocket, 318
 - window.navigator, 62
 - XMLHttpRequest, 98, 139, 318
 - zdarzeń, 170
- obraz bitmapowy, *Patrz:* grafika
bitmapowa
- okno aktywne, 24
- opcje zapisu gry, 24
- OpenGL, 24, 250, 251, 364

- OpenGL Shading Language, *Patrz:*
język GLSL
- Opera, 41, 97, 194, 250, 318
- operacja Portera-Duffa, 135
- oprogramowanie pośredniczące, 334
- orientacja, 70, 162
- oświetlenie, 273
- Phonga, 275
- pikselowe, 276

P

- pamięć podręczna, 313, 314, 315
- manifest, 313
- para klucz-wartość, 300
- parametr
 - device-height, 51
 - device-width, 51, 52
 - href, 21
 - initial-scale, 52
 - src, 21
- pasek
 - adresów, 69
 - postępu, 31, 147, 149, 207, 234, 235
 - stanu, 68, 69
- pauza, 50
- per-pixel lighting, *Patrz:*
oświetlenie pikselowe
- Phantom Limb, 174
- PhoneGap, 47, 338, 339, 340, 344
- PhoneGap Build, 340
- Phonga model, 273, 275
- plaszka, 32, 154, 155, 162
- kontener, 154
- uaktualnianie, 188
- Playtomic, 345
- plik cookie, 24, 298, 302, 318
- plaszczyna Béziera, 119
- plótno, *Patrz:* canvas
- podścieżka, 115
- polyfill, *Patrz:* wypełnienie
- połączenie sieciowe, 318, 325
- trwałe, 318
- ustanawianie, 318
- zamykanie, 319
- zerwanie, 320
- PopCap Games, 30
- Portera-Duffa operacja, 135
- pracownik sieciowy, 80, 96, 97, 98, 103, 105
- błędy, 99
- dedykowany, 100, 101
- możliwości, 97
- ograniczenia, 96
- wątek, *Patrz:* wątek pracownika
- współdzielony, 100, 101
- precyzja zmiennej, 257

- procesor graficzny, 251, 252
- projekcja perspektywy, 267
- promise object, *Patrz:* obiekt
obiecany
- prostokąt, 116, 281, 355
- protokół
 - HTTP, 23, 24, 318, 321, 324
 - WebSocket, *Patrz:* WebSocket
- Przedrostek autorski, 163, 194
- przełładarka, 37
- mobilna, 35, 65, 70
- mobilnym, 345
- okno, 51
- osobista, 35
- stacjonarna, 345
- zachowanie domyślne, 70
- przezeń operacyjna, 51
- przewijanie strony, 68
- przezroczystość, 116, 135, 136
- przycisk wirtualny, 47
- punkt ogniskowania, 263
- Puzzle Quest, 27

R

- ramka, 377
- ray tracer, *Patrz:* śledzenie
promieni
- referencja, 39
- renderowanie, 262, 265, 268, 269, 270, 286, 288, 376
- retained mode, *Patrz:* tryb
zatrzymany
- rozdzielczość, 35, 49, 57, 59, 66
- natywna, 49

S

- Safari, 41, 51, 61, 62, 68, 69, 70, 97, 100, 136, 193, 250, 318
- debugowanie, 71
- Scalable Vector Graphics, *Patrz:*
format SVG
- serwer
 - HTTP, 324
 - Linuksa, 324
 - wirtualny, 324
- shader, 259, 290, 366, 386
- fragmentów, 251, 252, 256, 257, 259, 261, 262, 276, 281, 282, 288, 290
- osadzanie, 258
- wierzchołków, 251, 252, 256, 257, 259, 262, 265, 267, 274, 275, 281
- shared worker, *Patrz:* pracownik
sieciowy współdzielony
- Sideshow, 40

silnik

Box2D, *Patrz:* Box2D
 efektów fizycznych, 334
 Impact, *Patrz:* Impact
 Three.js, *Patrz:* Three.js
 V8, 346
 Silverlight, 25, 27
 Sizzle, 33
 skalowanie, 52, 125, 162, 172, 192
 dynamiczne, 173
 skrypt, 36, 97, 98, 139
 słowo klucz, 73, 116, 176, 177
 bevel, 122
 browser, 73
 butt, 121
 device-width, 52
 em, 50
 inline-clock, 61
 miter, 122
 no-repeat, 124
 precision, 258
 repeat, 124
 repeat-x, 124
 repeat-y, 124
 round, 122
 serverConfig, 326
 source-over, 136
 square, 121
 uniform, 261
 słuchacz zdarzeń, 55
 smartfon, 49, 58, 60
 SmartMachines, 346
 Snake, 46
 socket, *Patrz:* gniazdo
 spacja em, 50
 specular light, *Patrz:* światło
 odbicia zwierciadlanego
 sprite, 144, 160, 175
 spyware, 24
 standard OpenGL, 250, *Patrz:*
 OpenGL
 state stack, *Patrz:* stos stanów
 stos stanów, 112, 130, 353
 styl
 obramowania, 117, 121, 124
 wypełniania, 117, 121, 124
 SVG, 20
 Symbian, 339
 symulacja zjawisk fizycznych, 104
 system
 dynamicznego wczytywania, 36
 iOS, *Patrz:* iOS
 logów, 73
 przydzielania zdarzeń, 80
 Ubuntu, *Patrz:* Ubuntu
 Szablewski Dominik, 335
 sztuczna inteligencja, 103

Ś

ścieżka, 115, 175, 353, 354
 dźwiękowa, 230, 244
 kontur, 115
 krzywa Béziera, *Patrz:* krzywa
 Béziera
 łuk, 117
 przecinająca się, 27
 przekształcanie na grafikę, 115
 przycinająca, 120
 przycinanie, 117
 wypełnienie, 115, 175
 śledzenie promieni, 113
 światło
 odbicia zwierciadlanego, 273, 277
 otaczające, 273, 274
 rozproszenia, 273, 274, 276

T

tablet, 49, 58, 168, 171
 tablica, 148
 bufora ramki, 237
 changedTouches, 171
 Float32Array, 237
 próbek, 236
 przeplatanych danych próbek,
 237
 targetTouches, 171, 173
 touches, 171
 typ, 365
 typowana, 237
 TapJS, 345
 tekstura, 153, 273, 278, 281, 290,
 371
 Three.js, 337
 Titanium, 344
 Titanium Studio, 341
 tło, 35, 151, 162
 półprzezroczyste, 26, 155
 transformacja, 353, *Patrz:* canvas
 transformacja
 transformata, 117
 transformation matrix, *Patrz:*
 macierz transformacji
 treść awaryjna, 114, 128
 trójkąt, 251, 262, 264, 271
 tryb
 asynchroniczny, 96, 322
 awaryjny, 227
 bezpieczeństwa, 319
 natychmiastowy, 113
 niezabezpieczony, 319
 pejzażowy, 58, 59, 60, 67, 208
 portretowy, 58, 68, 208
 synchroniczny, 96, 97
 zatrzymany, 113

TTF, *Patrz:* font TrueType
 typ medialny, 57

U

Ubuntu, 324
 ulepszenie
 progresywne, 26
 uniform variable, *Patrz:* zmienna
 jednolita
 URI, *Patrz:* identyfikator URI,
 Patrz: identyfikator URI
 urządzenie
 mobilne, 51, 59, 168, 235, 250,
 318, 338
 przenośne, 46
 rejestrujące dotyk, 56
 user agent, *Patrz:* klient
 użytkownika
 usługa sieciowa, 345
 użytkownika nazwa, 32

V

varying variable, *Patrz:* zmienna
 różnorodna
 vendor prefix, *Patrz:* przedrostek
 autorski
 Verou Lea, 163
 Vodori, 174

W

W3C, 20, 40, 174, 193, 235, 298,
 318, 361
 wątek, 96
 pracownika, 96, 99
 Web Font Working Group, 40
 Web Hypertext Application
 Technology Working Group,
 Patrz: WHATWG
 Web Open Font Format, *Patrz:*
 format WOFF
 Web Storage, *Patrz:* magazyn
 sieciowy
 web worker, *Patrz:* pracownik
 webgl, 251
 WebGL, 24, 25, 250, 251, 265,
 270, 284, 337, 364, 386
 WebKit, 23, 136, 163, 194
 WebOS, 339
 WebSocket, 23, 33, 63, 98, 318,
 322, 326, 346
 komunikacja, 321
 Weiner Ben, 40
 wektor, 253, 262, 265, 390, 391
 kierunku odbicia, 277
 mieszanie, 255
 widoku, 277

Weltmeister, 336
 WHATWG, 20, 28, 314
 wierzchołek, 369
 Windows, 341
 Windows Phone 7, 339
 właściwość medialna, 57
 WOFF, *Patrz:* format WOFF
 worker thread, *Patrz:* wątek
 pracownika
 współbieżność, 96
 wtyczka Flash, 22
 wynik, 32, 309
 rekordowy, 32, 309, 311, 312
 wypełnienie, 24, 27, 97
 wyrażenie regularne, 146
 wywołanie zwrotne, 322
 wzorzec modułów, 38

X

XHTML, 20
 XML, 20, 271
 XMLHttpRequest, 23

Y

yeppope, 36, 109, 146

Z

zapytanie medialne
 CSS, 46, 208
 CSS3, 57, 59
 zdarzenie, 113
 audio, 233
 click, 54, 55, 56, 168, 177
 connect, 327
 connection, 325
 delegacja, *Patrz:* delegacja
 zdarzenia
 dotykowe, 168, 170, 174, 178
 ended, 244
 gestów, 173, 174
 gesturestart, 173
 handler, 113
 keydown, 168, 179
 keypress, 168, 179
 keyup, 168, 179
 komunikacyjne globalne, 101
 łączności, 101
 message, 322
 mousedown, 168, 177
 mouseover, 168
 mouseup, 168
 MozAudioAvailable, 236
 nasłuchiwanie, 101
 płótna, 174
 sterowania, 174, 176, 177
 touch, 56
 touchend, 170
 touchmove, 68, 170, 173
 touchstart, 170, 171, 178
 wielodotykowe, 48, 171, 173
 Zeewe, 343
 zjawiska fizyczne, 334, 337
 w przestrzeni dwuwymiarowej,
 335, 337
 zmienna
 jednolita, 260, 265, 267, 275,
 281, 368
 różnorodna, 262, 281
 wektorowa, 261
 zmiennoprzecinkowa, 261, 262
 znacznik
 animowanie, 196
 canvas, 114
 embed, 226
 graficzny, 186
 meta, 34, 51, 62, 68, 69
 noscript, 114
 progress, 149

Ż

żądanie HTTP, 325

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>

GRUPA WYDAWNICZA

 **Helion SA**

Możliwości nowej odsłony języka HTML są nieprawdopodobne. Jego potencjał można wykorzystać do tworzenia atrakcyjnych stron WWW, funkcjonalnych aplikacji oraz... gier! Ten język świetnie sprawdza się nawet w takiej roli. WebGL, JavaScript oraz CSS3 pozwolą Ci rozwinąć skrzydła. Już dziś możesz podjąć wyzwanie i wraz z autorem tej wspaniałej książki stworzyć swoją pierwszą grę!

HTML5. Tworzenie gier napisano w przeświadczeniu, że książka pomoże Ci przenieść Twoje umiejętności tworzenia stron w świat projektowania gier komputerowych. Nie ma znaczenia, czy jesteś projektantem witryn, który pragnie spróbować swoich sił w nowej dziedzinie, twórcą aplikacji Flash zainteresowanym nową technologią, czy przyświeca Ci całkowicie inny cel. Książka pokaże Ci, jak użyć dobrze znanych narzędzi, by pokonać przepaść dzielącą tradycyjne witryny od ekscytujących gier. W trakcie lektury poznasz elementy składowe gry oraz charakterystykę urządzeń mobilnych. Zorientujesz się, jakie są kolejne etapy procesu wytwarzania nowej aplikacji. Nauczysz się tworzyć wielowątkowe rozwiązania dzięki technologii Web Workers oraz rysować atrakcyjne elementy graficzne przy użyciu elementu *canvas*. Z kolejnych rozdziałów dowiesz się, jak oprogramować sterowanie ruchem oraz jak przygotować efekty dźwiękowe dla Twojej gry, a potem poznasz tajniki tworzenia grafiki 3D oraz przechowywania danych lokalnie w przeglądarce użytkownika. Książka ta w sposób kompleksowy podchodzi do tematu tworzenia gier w języku HTML5. Jeżeli masz choć trochę zapału, dzięki niej z pewnością osiągniesz cel!

Już wkrótce będziesz mógł:

- zaprojektować swoją wyjątkową grę
- stworzyć zaawansowaną grafikę przy użyciu *canvas* i WebGL
- przechowywać dane lokalnie w bazie danych przeglądarki
- korzystać z wielowątkowości dzięki Web Workers

Od pustej strony do pełnowartościowej gry z HTML5!

Nr katalogowy: 10417



Księgarnia internetowa
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

- <http://helion.pl/promocje>
- Książki najchętniej czytane:
- <http://helion.pl/bestsellery>
- Zamów informacje o nowościach:
- <http://helion.pl/nowosci>

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

helion.pl
księgarnia
internetowa

Cena 67,00 zł

ISBN 978-83-246-4480-3



9 788324 644803

Informatyka w najlepszym wydaniu