



Technologia i rozwiązania

jQuery

Niezbędnik programisty gier

Tworzenie gier nie musi być trudne!

Helion



Selim Arsever

[PACKT]
PUBLISHING

Tytuł oryginału: jQuery Game Development Essentials

Tłumaczenie: Aleksander Lamża

ISBN: 978-83-246-8608-7

Copyright © Packt Publishing 2013.

First published in the English language under the title 'jQuery Game Development Essentials'.

Polish edition copyright © 2014 by Helion S.A.

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.

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/jqunpg>

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 autorze	9
O recenzencie	10
Wstęp	11
Co znajdziesz w książce	11
Co będzie Ci potrzebne	12
Dla kogo jest ta książka	12
Przyjęte konwencje	13
Kod przykładów	14
Rozdział 1. jQuery w grach	15
Jak używać biblioteki jQuery	15
Przesuwanie elementów	17
Obsługa zdarzeń	19
Wiązanie danych z elementami DOM	21
Manipulowanie strukturą DOM	21
Ciekawość to pierwszy stopień do...	23
Podsumowanie	23
Rozdział 2. Tworzymy pierwszą grę	25
Jak pracować z książką	26
Przejdźmy do konkretów — gra	26
Zaczynamy od podstaw	27
Inicjalizowanie gry	37
Główna pętla gry	39
Wykrywanie kolizji	42
Podsumowanie	45

Rozdział 3. Lepiej i szybciej, ale niekoniecznie trudniej	47
Interwały i odmierzanie czasu	48
Odpytywanie klawiatury	53
Fragmenty HTML	55
Unikanie przebudowywania struktury DOM	56
Przemieszczanie sprite'ów za pomocą przekształceń CSS	57
Zastosowanie requestAnimationFrame zamiast interwałów	58
Podsumowanie	59
Rozdział 4. Wskakujemy na głębszą wodę	61
Odłączane elementy div	61
Grupy	63
Przekształcenia sprite'ów	64
Mapy kafelków	68
Wykrywanie kolizji	71
Piszemy kod gry	75
Podsumowanie	85
Rozdział 5. Zmieniamy perspektywę	87
Optymalizowanie mapy kafelków dla gier z perspektywą mapy	88
Sortowanie przesłaniania	94
Wykrywanie kolizji	96
Kompletna gra	103
Izometryczne kafelki	104
Podsumowanie	105
Rozdział 6. Dodajemy kolejne poziomy	107
Implementowanie gry złożonej z wielu plików	108
Modyfikujemy grę platformową	117
Podsumowanie	121
Rozdział 7. Tworzymy grę typu multiplayer	123
World of Ar'PiGi	124
Zarządzanie kontem gracza	124
Synchronizacja graczy	132
Sterowanie przeciwnikami	137
Podsumowanie	141
Rozdział 8. Wkraczamy w sieci społecznościowe	143
Tworzenie prostej tablicy wyników	144
Mechanizmy utrudniające oszukiwanie	149
Integracja z Twitterem	156
Integracja z Facebookiem	164
Podsumowanie	171

Rozdział 9. Tworzymy grę mobilną	173
Jak sprawić, by gra dobrze działała na urządzeniach mobilnych?	174
Sterowanie dotykiem	180
Integracja gry z ekranem domowym	188
Korzystanie z informacji o orientacji urządzenia	191
Korzystanie z trybu offline	192
Lokalne składowanie danych	193
Podsumowanie	193
Rozdział 10. Ujarzmiamy dźwięk	195
Abstrakcyjna biblioteka obsługi dźwięku	196
Osadzanie dźwięku	198
Element audio	200
Web Audio API	204
Zastosowanie Flasha	211
Generowanie efektów dźwiękowych	214
Podsumowanie	214
Skorowidz	215

Wkraczamy w sieci społecznościowe

Już w najwcześniejszych grach wideo było stosowane pewne rozwiązanie zapewniające niesłabnące zainteresowanie graczy. Chodzi o **tablicę wyników**. Dzięki niej gracz stara się osiągnąć za każdym razem lepszy rezultat — od swojego poprzedniego wyniku, osiągnięć przyjaciół czy innych graczy z całego świata.

Sieci społecznościowe nadały nowy wymiar temu prostemu pomysłowi, ponieważ osiągnięty rezultat można opublikować na osi czasu albo w tweecie. Ma to wiele zalet, a jedną z nich jest z całą pewnością to, że jeżeli ktoś zamieści swój wynik, jego znajomi z dużym prawdopodobieństwem będą również chcieli zagrać w tę grę (i osiągnąć lepszy rezultat!).

W tym rozdziale pracę rozpoczniemy od przygotowania skryptu tablicy wyników działającego po stronie serwera. Zastosujemy tu techniki znane z poprzedniego rozdziału. Następnie zajmiemy się umożliwieniem zalogowania się do gry za pomocą konta na Twitterze oraz umieszczeniem tweetów z wynikami.

Ostatnia część rozdziału jest poświęcona integracji gry z Facebookiem, publikowaniu wydarzeń na osi czasu użytkownika oraz tworzeniu osiągnięć.

Podczas korzystania z serwisów takich jak Facebook czy Twitter trzeba zwracać szczególną uwagę, by postępować zgodnie z ustalonymi tam zasadami oraz na bieżąco śledzić wszelkie aktualizacje, dzięki czemu gra będzie zawsze poprawnie funkcjonowała. Aplikacje i gry, które nie spełniają wymagań, są zazwyczaj blokowane.

Dowiesz się, jak korzystać z tych dwóch konkretnych sieci społecznościowych, ale musisz wiedzieć, że mechanizm stojący za większością innych tego typu serwisów jest podobny.

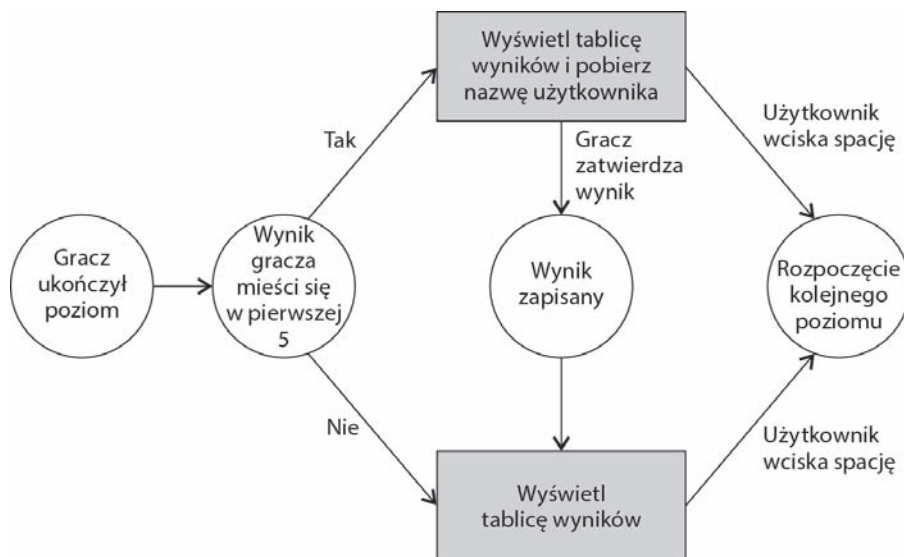
W tym rozdziale zostaną omówione następujące zagadnienia:

- tworzenie prostej tablicy wyników,
- wprowadzanie mechanizmów utrudniających oszukiwanie,
- integracja gry z Twitterem umożliwiającą publikowanie wyników,
- integracja gry z Facebookiem pozwalająca na zdobywanie osiągnięć.

Tworzenie prostej tablicy wyników

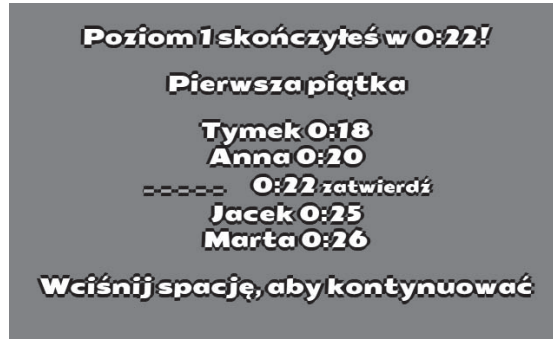
Jak się można domyślić, utworzenie tablicy wyników będzie wymagało bazy danych, w której będzie przechowywana punktacja. Podobnie jak w poprzednim rozdziale, do implementacji gry po stronie serwera użyjemy języka PHP i bazy danych MySQL. Jednak w przeciwieństwie do problemu omówionego w poprzednim rozdziale, realizacja tablicy wyników dla wielu graczy jest o wiele prostsza. Wynika to stąd, że nie wymaga zbyt wielu zasobów serwera, a operacje na bazie danych nie są wykonywane zbyt często. Można założyć, że dla każdego gracza żądanie będzie zgłaszane co mniej więcej 10 sekund, a nie — jak w przypadku gry MMORPG z poprzedniego rozdziału — kilka razy w ciągu sekundy.

Przede wszystkim musimy ustalić, co będzie pełniło funkcję punktacji w grze. Zastosujemy najprostsze rozwiązanie, czyli zdecydujemy się na czas (wyrażony w sekundach), w którym gracz przeszedł planszę. Poniższy diagram ilustruje algorytm implementowanego mechanizmu:



Na potrzeby interfejsu użytkownika będziemy musieli utworzyć dwa ekrany. Postąpimy tak samo jak w poprzednim rozdziale, czyli użyjemy elementów `div`, które będziemy wyświetlać lub ukrywać.

Pierwszy ekran służy do poinformowania o rozpoczęciu poziomu i daje graczowi czas na przygotowanie się. Drugi ekran jest trochę bardziej złożony. Wyświetlane są na nim: uzyskany wynik, lista pięciu najlepszych graczy oraz, jeśli aktualny wynik gracza kwalifikuje się do listy najlepszych, pole umożliwiające wpisanie się na nią. Przykładowy wygląd tego ekranu został przedstawiony poniżej:



Zdecydowaliśmy się na takie rozwiązanie — pytanie o nazwę użytkownika nie na początku gry, ale dopiero po uzyskaniu wystarczającej liczby punktów — by odtworzyć zachowanie klasycznych gier.

Podsumowując, serwer musi obsłużyć dwie dodatkowe operacje:

- pobranie pięciu najlepszych wyników dla danego poziomu,
- zapisanie punktacji dla danego poziomu.

Zaimplementujemy je w osobnych plikach: *highscore.php* oraz *save.php*.

Zapisywanie wyników

Tabela w bazie danych musi się składać z trzech kolumn:

- `level` — liczba całkowita określająca numer poziomu gry,
- `name` — łańcuch znaków przechowujący nazwę użytkownika,
- `time` — liczba całkowita odpowiadająca liczbie sekund, która upłynęła od rozpoczęcia poziomu do jego zakończenia.

Skrypt zapisujący wyniki jest bardzo prosty. Wystarczy, że na serwer prześlemy nazwę użytkownika, liczbę punktów oraz poziom. Otrzymane dane zapiszemy w bazie danych za pomocą następującej kwerendy:

```
INSERT INTO scores (level, name, time) VALUES (1, "Krysią", 36);
```

Pozostała część skryptu jest bardzo podobna do tego, z czym mieliśmy do czynienia w poprzednich rozdziałach, więc nie będziemy tu zamieszczać kodu. Pełną wersję skryptu znajdziesz w plikach dołączonych do książki.

Pobieranie listy najlepszych wyników

Aby pobrać wyniki z serwera, wystarczy przekazać mu numer poziomu, a on zwróci punktację. Zastosowaliśmy tu jednak trochę bardziej skomplikowany mechanizm. Ustalamy dodatkowo, czy bieżący użytkownik zakwalifikował się na listę, i — jeśli tak — obliczamy, na którym miejscu. Dzięki temu będziemy mogli również zaimplementować zabezpieczenia przeciwdziałające oszustwom.

Musimy więc dostarczyć do serwera poziom oraz uzyskany czas, a on zwróci plik JSON zawierający wszystkie dane niezbędne do wygenerowania tablicy wyników. Format tego pliku przedstawia się następująco:

```
{
  "top":[
    {"name": "Tymek", "time": 18},
    {"name": "Anna", "time": 20},
    {"time": 22},
    {"name": "Jacek", "time": 25}
  ],
  "intop": true,
  "pos": 2
}
```

Jak można zauważyć, wprowadziliśmy pole `intop`, które informuje, czy bieżący użytkownik znalazł się na liście pięciu najlepszych. Jeśli ma ono wartość `true`, dołączane jest kolejne pole — `pos`, które przechowuje indeks rezultatu gracza w tablicy wyników (`top`). Wszystkie pozostałe elementy tablicy `top` odpowiadają uporządkowanym rosnąco wynikom uzyskanym przez innych graczy. Jeśli pole `intop` ma wartość `false`, tablica `top` przechowuje jedynie wyniki innych graczy.

Pierwszym krokiem będzie wykonanie poniższego zapytania SQL:

```
SELECT * FROM scores WHERE level=1 ORDER BY time ASC LIMIT 5;
```

Jest ono bardzo podobne do poprzednich, ale na końcu (fragment zapisany wyróżnionym kodem) znajduje się modyfikator wymuszający posortowanie wyników względem kolumny `time` w porządku rosnącym (`ORDER BY time ASC`) oraz ograniczający liczbę zwracanych rekordów do pięciu (`LIMIT 5`).

W drugiej kolejności trzeba się zająć wygenerowaniem pliku JSON na podstawie danych otrzymanych z kwerendy. Jedynym ciekawym fragmentem jest wstawienie wyniku gracza, jeśli uzyskał wystarczająco dobry czas. Pełny kod tego skryptu znajduje się poniżej:

```
<?php
  session_start();

  include 'dbconnect.php';

  $time = $_GET['time'];
```

```

$level = $_GET['level'];

if (isset($time) && isset($level)) {
    // Obiekt JSON
    $json = array('top'=>array(), 'intop'=>false);

    $query = 'SELECT * FROM scores WHERE level='.$level.' ORDER BY time ASC
↳LIMIT 5';
    $result = mysqli_query($link, $query);
    $i=0;

    while ($obj = mysqli_fetch_object($result)) {
        if (!$json['intop'] && $time < $obj->time) {
            $json['intop'] = true;
            $json['pos'] = $i;

            array_push($json['top'], array('time'=>$time));

            $i++;
        }
        if ($i < 5) {
            array_push($json['top'], array('time'=>$obj->time, 'name'=>$obj-
↳>name));
            $i++;
        }
    }

    if ($i < 5 && !$json['intop']) {
        $json['intop'] = true;
        $json['pos'] = $i;

        array_push($json['top'], array('time'=>$time));
    }

    mysqli_free_result($result);
    echo json_encode($json);
}
mysqli_close($link);
?>

```

Na listingu zostały wyróżnione te fragmenty, które odpowiadają za operacje na wyniku gracza.

Wyświetlanie listy najlepszych wyników

Po stronie klienta wygenerujemy ekran przedstawiający wynik, listę najlepszych graczy oraz pole służące do wprowadzenia nazwy użytkownika, jeśli udało mu się uzyskać wystarczająco dobry czas. Kod ten przedstawia się następująco:

```

var finishedTime = Math.round((Date.now() - levelStart) / 1000);
$.ajax({
  dataType: "json",
  url: "highscore.php",
  data: {
    level: currentLevel,
    time: finishedTime
  },
  async: false,
  success: function(json) {
    var top = "";
    for (var i = 0; i < json.top.length; i++) {
      if (json.intop && json.pos === i) {
        top += "<input id='name' placeholder='____' size='5' />" +
          ↵ "<input id='timeScore' type='hidden' value='"+json.top[i]
          ↵ ".time+'></input>" + "<input id='level' type='hidden' value="
          ↵ "+currentLevel+'></input>" + " "+minSec(json.top[i].time) + "
          ↵ "<a id='saveScore' href='#'>submit</a> <br>";
      } else {
        top += "" + json.top[i].name + " " + minSec(json.top[i].time) +
          ↵ "<br>";
      }
    }
    $("#top_list").html(top);
  }
}).fail(function(a, b, c) {
  var toto = "toto";
});

```

Kod generujący listę został wyróżniony. Tworzymy pole, w którym gracz może wpisać swoją nazwę, oraz dwa ukryte pola przechowujące numer poziomu i punktację. Umieszczamy tu również odsyłacz służący do zatwierdzenia wyniku. Kod obsługujący ten odsyłacz znajduje się poniżej:

```

$("#levelEnd").on("click", "#saveScore", function() {
  $.get("save.php", {
    name: $("#name").val(),
    time: $("#timeScore").val(),
    level: $("#level").val()
  }, function() {
    $("#saveScore").fadeOut(500);
  });
  return false;
});

```

Pobieramy tu wartości z pól i przekazujemy je na serwer. Po ich przesłaniu usuwamy przycisk zatwierdzania, aby dać znać użytkownikowi, że proces się zakończył.

Mechanizmy utrudniające oszukiwanie

Nie istnieje idealne rozwiązanie uniemożliwiające oszukiwanie. Problem ten jest szczególnie istotny w grach tworzonych w JavaScriptcie, ponieważ bardzo łatwo uzyskać dostęp do kodu źródłowego. Można oczywiście przekształcić kod do nieczytelnej postaci, ale to nie powstrzyma zdeterminowanego oszusta, a jedynie spowolni jego pracę. Istnieje jednak kilka innych technik, dzięki którym próby oszustw stają się trudniejsze i mniej opłacalne.

Weryfikacja po stronie serwera

Najprostszym sposobem na utrudnienie oszukiwania jest przeniesienie części kodu na serwer. Zrobiliśmy tak z mechanizmem walki w grze tworzonyj w rozdziale 7. W przypadku gry platformowej musielibyśmy przysyłać na serwer informacje o wciskaniu klawiszy sterujących postacią, na podstawie których skrypt określałby jego nowe położenie.

W większości przypadków nie da się zastosować takiego rozwiązania w tego typu grach, ale można stworzyć skrypt, który będzie weryfikował punktację gracza po stronie serwera. Wystarczy w kilku miejscach planszy umieścić niewidoczne punkty kontrolne, po osiągnięciu których na serwer jest przesyłana stosowna informacja. Dzięki temu, jeśli użytkownik przesłał na serwer punktację, ale nie trafiły tam wcześniej informacje z punktów kontrolnych, wiemy, że coś jest nie tak. Można też zastosować podobne rozwiązanie, ale bazujące na przykład na liczbie skoków lub porażek gracza.

Tak czy inaczej, trzeba wprowadzić w grze jakąkolwiek formę weryfikacji wyników. Nie ma jednego, najlepszego rozwiązania. Musisz jednak pamiętać, by zastosowany mechanizm nie wykrył próby oszustwa tam, gdzie go nie ma, czyli żeby uczciwy gracz nie został o nie posądzony. Istotny jest również nakład pracy wymagany do zaimplementowania tego typu mechanizmów, tak by praca nad nimi nie zajęła przypadkiem więcej czasu niż tworzenie samej gry.

W przedstawionym poniżej przykładzie zastosujemy proste rozwiązanie. Znamy maksymalną prędkość, z jaką może się poruszać gracz, wiemy też, ile ma do przejścia, więc łatwo jest obliczyć minimalny czas potrzebny na zakończenie poziomu. Wystarczy porównać wynik gracza z tą wartością i sprawdzić, czy nie jest mniejszy.

Aby to zrobić, trzeba uzupełnić skrypt *highscore.php*:

```
// gracz może przejść 7 pikseli w 30 ms -> 233.1
$minTime = array(
    1 => 15, // 3500 / 233.1
    2 => 15, // 3500 / 233.1
    3 => 42, // 9800 / 233.1
    4 => 23 // 5460 / 233.1
);
$timeValid = !($minTime[intval($level)] < intval($time));
//...
while ($obj = mysqli_fetch_object($result)) {
```

```

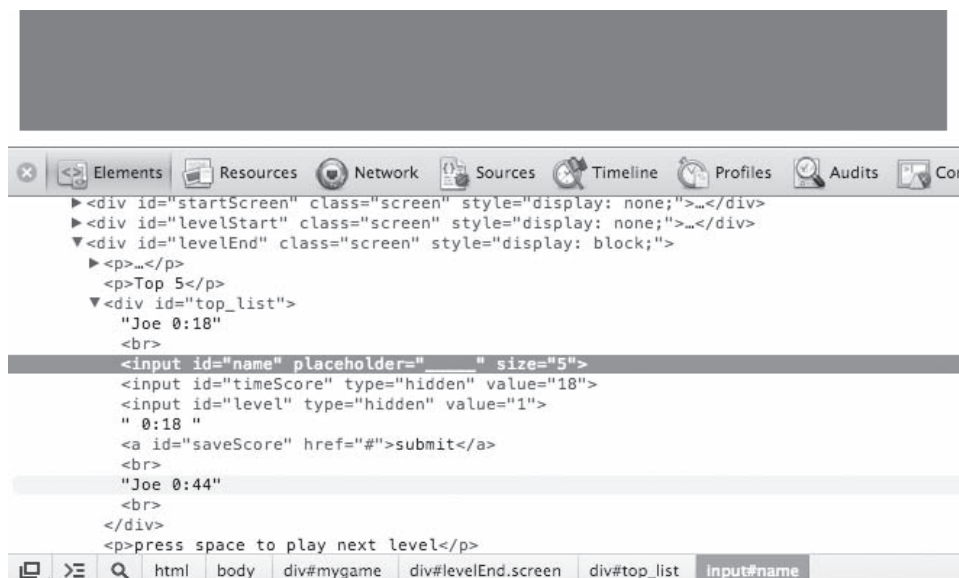
    if (!$json['intop'] && $time < $obj->time && $timeValid) {
        //...
    }

```

Jeśli wynik osiągnięty przez gracza został uznany za nieprawdziwy, zostanie wyświetlony, ale gracz nie będzie mógł wpisać swojej nazwy.

Mniej czytelne zmienne

Innym rozwiązaniem jest zmniejszenie czytelności zmiennych, tak by utrudnić analizę danych przesyłanych na serwer oraz ich modyfikację (dane te są w naszym przypadku zapisane w ukrytych polach formularza). Rozwiązanie to wydaje się dobre, ale niestety nietrudno jest obejść takie zabezpieczenia. Na poniższym rysunku zostało przedstawione okno inspektora przeglądarki Chrome.



Jedną z naczelnych zasad jest unikanie przechowywania istotnych informacji w elementach znajdujących się w strukturze DOM, ponieważ mają do nich dostęp wszyscy użytkownicy, nawet tacy, którzy z programowaniem mają niewiele do czynienia. W związku z tym usuniemy je z wywołania skryptu *save.php* i zapiszemy je w sesji. W pliku *highscore.php* musimy dodać następujący kod:

```

if (!$json['intop'] && $time < $obj->time && $timeValid) {
    $json['intop'] = true;
    $json['pos'] = $i;

    array_push($json['top'], array('time'=>$time));
}

```

```

    $_SESSION['level'] = $level;
    $_SESSION['time'] = $time;

    $i++;
}

```

W skrypcie *save.php* musimy z sesji odczytać poziom i czas:

```

$name = $_GET['name'];
$time = $_SESSION['time'];
$level = $_SESSION['level'];

```

Te proste zmiany wystarczą, by utrudnić oszukiwanie w naszej grze.

Zaciemnianie kodu

Zaciemnianie kodu (ang. *obfuscating*) jest bardzo prostą techniką, ale przy tym całkiem skuteczną. Przekształcony w ten sposób kod trudno analizować, ponieważ jest nieczytelny dla ludzi. Poniżej znajduje się fragment kodu odpowiedzialnego za tablicę wyników:

```

if (status == "finished") {
    gameState = "menu";
    $("#level_nb_2").html(currentLevel);
    $("#level_nb_1").html(currentLevel + 1);

    var finishedTime = Math.round((Date.now() - levelStart) / 1000);
    $.ajax({
        dataType: "json",
        url: "highscore.php",
        data: {
            level: currentLevel,
            time: finishedTime
        },
        async: false,
        success: function (json) {
            var top = "";
            for (var i = 0; i < json.top.length; i++) {
                if (json.intop && json.pos === i) {
                    top += "<input id='name' placeholder='____' size='5' />" +
                        " <input id='timeScore' type='hidden' value='"+json.top[i].
                        time+"'></input>" + "<input id='level' type='hidden' value='"
                        +currentLevel+"'></input>" + " "+minSec(json.top[i].time) + "
                        <a id='saveScore' href='#'>submit</a> <br>";
                } else {
                    top += " " + json.top[i].name + " " + minSec(json.top[i].time) +
                        "<br>";
                }
            }
        }
    });
}

```

```

    }
    $("#top_list").html(top);
  }
}).fail(function(a, b, c) {
  var toto = "toto";
});

$("#time").html(minSec(finishedTime));

$("#levelEnd").fadeIn(2000, function() {
  $("#backgroundFront").css("background-position", "0px 0px");
  $("#backgroundBack").css("background-position", "0px 0px");
  gf.x(group, 0);

  tilemap = loadNextLevel(group);
  gf.x(player.div, 0);
  gf.y(player.div, 0);
  gf.setAnimation(player.div, playerAnim.jump);
});
status = "stand";
}

```

Ten sam kod poddany operacji zaciemniania (za pomocą narzędzia UglifyJS) wygląda następująco:

```

if("finished"==status){gameState="menu",$("#level_nb_2").html(currentLevel),$("#level_nb_1").html(currentLevel+1);var finishedTime=Math.round((Date.now()-levelStart)/1e3);$.ajax({dataType:"json",url:"highscore.php",data:{level:currentLevel,time:finishedTime},async:!1,success:function(a){for(var b="",c=0;a.top.length>c;c++)b+=a.intop&&a.pos===c?"<input id='name' placeholder='_____' size='5' /><input id='timeScore' type='hidden' value='"+a.top[c].time+"'></input>"+<input id='level' type='hidden' value='"+currentLevel+"'></input>"+ " "+minSec(a.top[c].time)+" <a id='saveScore' href='#!'>submit</a> <br>":" "+a.top[c].name+" "+minSec(a.top[c].time)+"<br>";$("#top_list").html(b)}}).fail(function(){}),$("#time").html(minSec(finishedTime)),$("#levelEnd").fadeIn(2e3,function(){$("#backgroundFront").css("background-position","0px 0px"),$("#backgroundBack").css("background-position","0px 0px"),gf.x(group,0),tilemap=loadNextLevel(group),gf.x(player.div,0),gf.y(player.div,0),gf.setAnimation(player.div,playerAnim.jump)}),status="stand"}

```

Z całą pewnością analiza takiego kodu jest dużo trudniejsza, a zyskujemy przy okazji coś jeszcze — przetworzony w ten sposób kod jest krótszy niż oryginał.

Mniej czytelny protokół sieciowy

Chociaż poprawiliśmy kod działający po stronie klienta, nadal istnieje możliwość uzyskania dostępu do wartości przesyłanych zmiennych poprzez analizę ruchu sieciowego. Na poniższym rysunku zostało przedstawione okno aplikacji nasłuchującej, która wyświetla dane przesyłane po zakończeniu poziomu.

```

6 0.000548000 localhost localhost TCP 76 ddi-tcp-1 > 50241 [f
7 0.006098000 localhost localhost HTTP 472 HTTP/1.1 200 OK (te
8 0.006140000 localhost localhost TCP 76 50241 > ddi-tcp-1 [f
▶ Frame 5: 631 bytes on wire (5048 bits), 631 bytes captured (5048 bits) on interface 0
▶ Null/Loopback
▶ Internet Protocol Version 6, Src: localhost (::1), Dst: localhost (::1)
▶ Transmission Control Protocol, Src Port: 50241 (50241), Dst Port: ddi-tcp-1 (8888), Seq: 1, Ack: 1, Len: 555
▼ Hypertext Transfer Protocol
  ▶ GET /book/code/chapter%208/save.php?name=Test&time=18&level=1 HTTP/1.1\r\n
    Host: localhost:8888\r\n
    Connection: keep-alive\r\n
    Accept: */*\r\n
  X-Requested-With: XMLHttpRequest\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.
  00c0 0d 0a 41 63 63 65 70 74 3a 20 2a 2f 2a 0d 0a 58 ..Accept: */*..
00d0 2d 52 65 71 75 65 73 74 65 64 20 57 69 74 69 3a ..Requested-With:
00e0 20 63 4d 4c 49 74 71 7d 52 65 71 75 65 73 74 6d XMLHttpRequest
00f0 6a 55 73 65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a User-Agent: Moz
0100 69 6c 6c 61 2f 35 2e 30 20 28 4d 61 63 69 6e 74 illa/5.0 (Macint
0110 6f 73 68 3b 20 49 6e 74 65 6c 20 4d 61 63 20 4f osh; Intel Mac OS
0120 53 20 58 20 31 30 5f 38 5f 32 29 20 41 70 70 6c S X 10_8_2) Appl
0130 65 57 65 62 4b 69 74 2f 35 33 37 2e 31 37 20 28 ewebKit/537.17 (
0140 4b 48 54 4d 4c 2c 20 6c 69 6b 65 20 47 65 63 6b KHTML, like Geck
0150 6f 29 20 43 68 72 6f 6d 65 2f 32 34 2e 30 2e 31 o) Chrome/24.0.1
0160 22 21 27 2e 2e 27 20 52 61 66 61 70 60 2f 2e 22 21 27 2e 2e 27 20 52 61 66 61 70 60 2f 2e 22

```

Jest to problem, ponieważ w celu oszustwa użytkownik nie musi modyfikować kodu — wystarczy sfalszować pakiet przesyłany na serwer. Istnieją trzy proste sposoby na utrudnienie analizowania transmitowanych danych:

1. Nadawanie zmiennym przypadkowych nazw, tak by na ich podstawie nie dało się określić ich przeznaczenia.
2. Kodowanie zawartości zmiennych. Użytkownik zna przeważnie osiągnięty przez siebie wynik, więc na jego podstawie może znaleźć w transmitowanych danych odpowiednią zmienną i ją zmodyfikować. Dzięki zakodowaniu tej wartości zadanie to stanie się trudniejsze.
3. Dodanie wielu przypadkowych zmiennych. Dzięki temu trudniej się domyślić, która z nich przechowuje istotne dane.

Podobnie jak w przypadku wcześniej opisywanych metod, te również jedynie utrudniają próby oszustwa, ale nie dają stuprocentowego zabezpieczenia. Równoczesne zastosowanie kilku rozwiązań może zniechęcić nawet najbardziej zdeterminowanych oszustów. Poniżej znajdziemy propozycję implementacji każdego z nich.

Kodowanie wartości

Zacniemy od kodowania wartości. Można to zrobić na wiele sposobów różniących się używanym stopniem bezpieczeństwa. W naszym przypadku wystarczy utrudnić odszukanie wyniku na liście przesyłanych zmiennych, w związku z czym nie musimy wprowadzać żadnej

skomplikowanej metody kodowania. W kodzie klienta użyjemy operacji przesunięcia w lewo (<<), a na serwerze — przesunięcia w prawo (>>).

Oto kod skryptu działającego na kliencie:

```
$.ajax({
  dataType: "json",
  url: "highscore.php",
  data: {
    level: currentLevel,
    time: finishedTime << 1
  },
  async: false,
  success: function (json) {
    //...
  }
});
```

Odpowiedni fragment skryptu na serwerze:

```
$time = intval($_GET['time']) >> 1;
```

Aby jeszcze bardziej zmylić użytkownika, wartość tę prześlemy w wielu innych zmiennych, które nie będą w ogóle brane pod uwagę po stronie serwera.

Przypadkowe nazwy zmiennych

Nie trzeba tu wiele wyjaśniać — wystarczy zmienić nazwę zmiennej. Jeśli chcesz jeszcze bardziej utrudnić zadanie oszustom, możesz zmieniać tę nazwę przy każdym żądaniu. Poniżej podstawowa wersja kodu:

```
$.ajax({
  dataType: "json",
  url: "highscore.php",
  data: {
    Nmyszf: currentLevel,
    WfBCLQ: finishedTime << 1
  },
  async: false,
  success: function (json) {
    //...
  }
});
```

Kod po stronie serwera przyjmie następującą postać:

```
$time = intval($_GET['WfBCLQ']) >> 1;
$level = $_GET['Nmyszf'];
```

Dodatkowe przypadkowe zmienne

Nazwy zmiennych nie oddają już znaczenia przechowywanych przez nie wartości, ale to nie wszystko, ponieważ stosunkowo łatwo sprawdzić, która z nich odpowiada za wynik. Musimy więc dodać więcej zmiennych, których zadaniem jest wprowadzenie oszusta w błąd. Przykładowe rozwiązanie zostało przedstawione na poniższym listingu:

```
$.ajax({
  dataType: "json",
  url: "highscore.php",
  data: {
    sXZUj: Math.round(200*Math.random()),
    enHf8F: Math.round(200*Math.random()),
    eZnqBG: currentLevel,
    avFanB: Math.round(200*Math.random()),
    zkpCfb: currentLevel,
    PCXFTR: Math.round(200*Math.random()),
    Nmyzsf: currentLevel,
    FYGswH: Math.round(200*Math.random()),
    C3kaTz: finishedTime << 1,
    gU7buf: finishedTime,
    ykN65g: Math.round(200*Math.random()),
    Q5jUZm: Math.round(200*Math.random()),
    bb5d7V: Math.round(200*Math.random()),
    WTsrdm: finishedTime << 1,
    bCW5Dg: currentLevel,
    AFM8MN: Math.round(200*Math.random()),
    FUht6K: Math.round(200*Math.random()),
    WfBCLQ: finishedTime << 1,
    d8mzVn: Math.round(200*Math.random()),
    bHxNpb: Math.round(200*Math.random()),
    MWcmCz: finishedTime,
    ZAat42: Math.round(200*Math.random())
  },
  async: false,
  success: function (json) {
    //...
  }
});
```

W kodzie serwera nie trzeba nic zmieniać, ponieważ dodatkowe zmienne są po prostu ignorowane. Można oczywiście jeszcze bardziej skomplikować sprawę i wprowadzić dodatkowe zależności między zmiennymi. Należy jednak cały czas pamiętać, które zmienne przechowują istotne dane, by również sobie nie utrudnić zadania.

Integracja z Twitterem

Twitter daje możliwość dzielenia się z innymi krótkimi informacjami. W przypadku gier możemy z niego skorzystać na dwa sposoby:

- możemy umożliwić graczowi zalogowanie się do gry za pomocą nazwy użytkownika i hasła z Twittera,
- możemy dać możliwość publikowania tweetów z wynikami lub informacjami o postępach w grze.

Zajmiemy się teraz zaimplementowaniem tych dwóch funkcji.

Podstawy Twittera

Istnieje bardzo prosty sposób korzystania z Twittera, który nie wymaga znajomości żadnego API. Jeśli użytkownik jest zalogowany, za pośrednictwem odpowiednio przygotowanego adresu URL można opublikować tweet. URL powinien być zbudowany w następujący sposób:

`http://twitter.com/home?status=Przykładowy tweet`

Wyróżniony fragment adresu to tekst tweeta. Na ekranie tablicy wyników obok przycisku *Zatwierdź* umieścimy odsyłacz *Tweetnij*.

```
$.ajax({
  dataType: "json",
  url: "highscore.php",
  data: {
    // ...
  },
  async: false,
  success: function (json) {
    var top = "";
    for (var i = 0; i < json.top.length; i++) {
      if (json.intop && json.pos === i) {
        top += "<input id='name' placeholder='_____' size='5' />" + "
        ↵"+minSec(json.top[i].time) + " <a id='saveScore' href='#'>submit
        ↵</a>" + " <a id='tweetScore' target='_blank' href='http://
        ↵twitter.com/home?status="+escape("Właśnie udało mi się skończyć
        ↵"+currentLevel+" poziom gry Yet Another Platformer w
        ↵"+minSec(json.top[i].time)+" sekund!"+'">Tweetnij</a> <br>";
      } else {
        top += " " + json.top[i].name + " " + minSec(json.top[i].time) +
        ↵"<br>";
      }
    }
    $("#top_list").html(top);
  }
});
```

Najistotniejszy jest wyróżniony fragment kodu, bo to dzięki niemu dzieje się to, co powinno. Zastosowaliśmy tu javascriptową funkcję `escape`, by dostarczany test był poprawnie sformatowany.

Jak widać, bardzo łatwo zaimplementować to rozwiązanie, ale trzeba wiedzieć o kilku ograniczeniach:

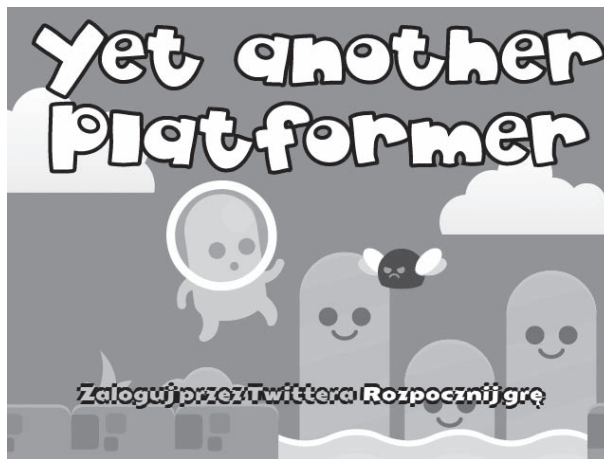
- Jeśli użytkownik nie jest zalogowany, pojawi się ekran logowania Twittera.
- Na ekranie tablicy wyników nie można użyć nazwy użytkownika z Twittera. W związku z tym, nawet jeśli użytkownik będzie chciał tweetnąć swój wynik, i tak będzie musiał podać swoją nazwę w grze.
- Publikowanie każdego tweeta powoduje otwarcie nowego okna przeglądarki, w którym użytkownik musi potwierdzić tę operację.

Jeśli chcielibyśmy umożliwić użytkownikowi zalogowanie się i automatyczne publikowanie tweetów bez konieczności każdorazowego otwierania nowego okna, musimy skorzystać z API Twittera.

Pełny dostęp do API Twittera

Najpełniejsza integracja gry z Twitterem wymaga zapytania użytkownika o zgodę na połączenie jego konta z grą. Można w tym celu skorzystać z mechanizmu OAuth, który jest otwartym standardem uwierzytelniania stosowanym przez wiele firm, takich jak Twitter, Google czy Facebook.

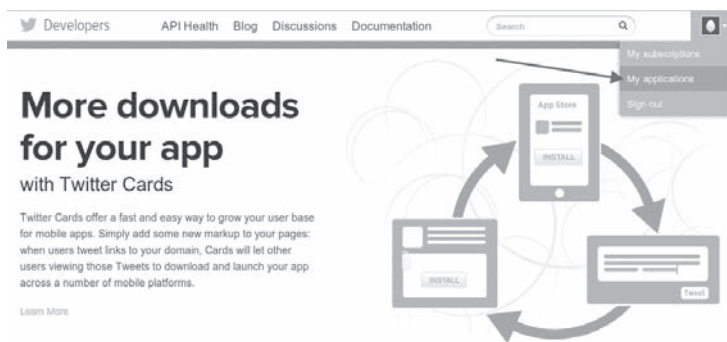
Aby dać użytkownikom możliwość zalogowania się za pośrednictwem Twittera, musimy lekko zmodyfikować ekran początkowy:



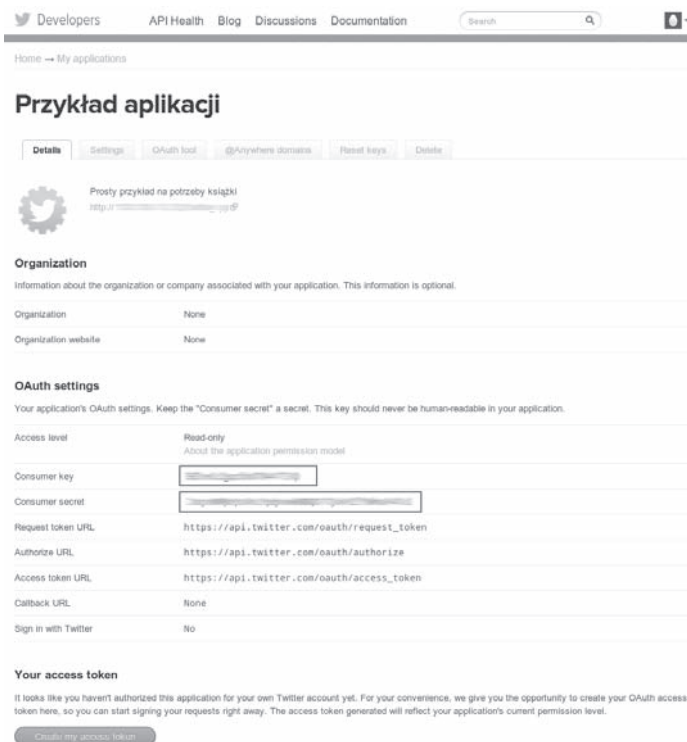
Jeśli gracz kliknie łącze *Rozpocznij grę*, zacznie od razu rozgrywkę. Jeśli z kolei kliknie *Zaloguj przez Twittera*, będzie się musiał zalogować za pośrednictwem Twittera, po czym nastąpi powrót do gry.

Rejestrowanie gry w Twitterze

Zanim przejdziemy dalej, musimy zarejestrować grę w Twitterze. Aby to zrobić, musisz się zalogować na stronę Twittera dla deweloperów (<https://dev.twitter.com>), a następnie kliknąć pozycję *My applications* (moje aplikacje) w menu profilu:



Należy utworzyć nową aplikację (łącze *Create a new application*), wypełnić wszystkie wymagane pola oraz wyrazić zgodę na warunki licencyjne (*Rules of the Road*). Po zatwierdzeniu wprowadzonych danych zostanie wyświetlona strona zawierająca dane utworzonej aplikacji:



Trzeba zwrócić uwagę na dwie informacje, które na rysunku zostały otoczone ramką, ponieważ będą potrzebne później. Należy wykonać jeszcze jedną drobną modyfikację konfiguracji. Przejdź do zakładki *Settings* (ustawienia) i przyjrzyj się sekcji *Application Type* (typ aplikacji). Domyślnie jest zaznaczone pole *Read only* (tylko do odczytu). Jeśli chcesz w imieniu gracza publikować tweety, musisz wybrać opcję *Read and Write*:

The screenshot shows the 'Application Details' and 'Application Type' sections of the Twitter Developer portal. In the 'Application Type' section, the 'Read and Write' option is selected, indicated by a red box and an arrow. Other sections include 'Name', 'Description', 'Website', 'Application Icon', and 'Callback URL'.

W ten sposób przygotowaliśmy konto na Twitterze do integracji z grą.

Pomocnicza biblioteka

Kod realizujący komunikację z API Twittera można napisać samemu w PHP, ale to dosyć żmudna praca. Na szczęście jest dostępnych wiele bibliotek, które mogą to uprościć. Jedną z nich, przeznaczoną dla PHP, jest **twitteroauth** (<https://github.com/abraham/twitteroauth>). Istnieją oczywiście biblioteki dla innych języków; koniecznie zajrzyj do dokumentacji Twittera, aby dowiedzieć się więcej na ten temat.

Dużą zaletą biblioteki **twitteroauth** jest to, że można ją zainstalować na prawie każdym serwerze obsługującym PHP. Wystarczy skopiować bibliotekę do tego samego katalogu, w którym znajdują się skrypty gry. W naszym przypadku będzie to podkatalog *twitter*.

Musimy zacząć od skonfigurowania biblioteki. Przydatny będzie przykładowy skrypt konfiguracyjny *config-sample.php* dostępny w podanym repozytorium. Znajduje się w nim kod definiujący trzy stałe:

```
define('CONSUMER_KEY', '(1)');
define('CONSUMER_SECRET', '(2)');
define('OAUTH_CALLBACK', '(3)');
```

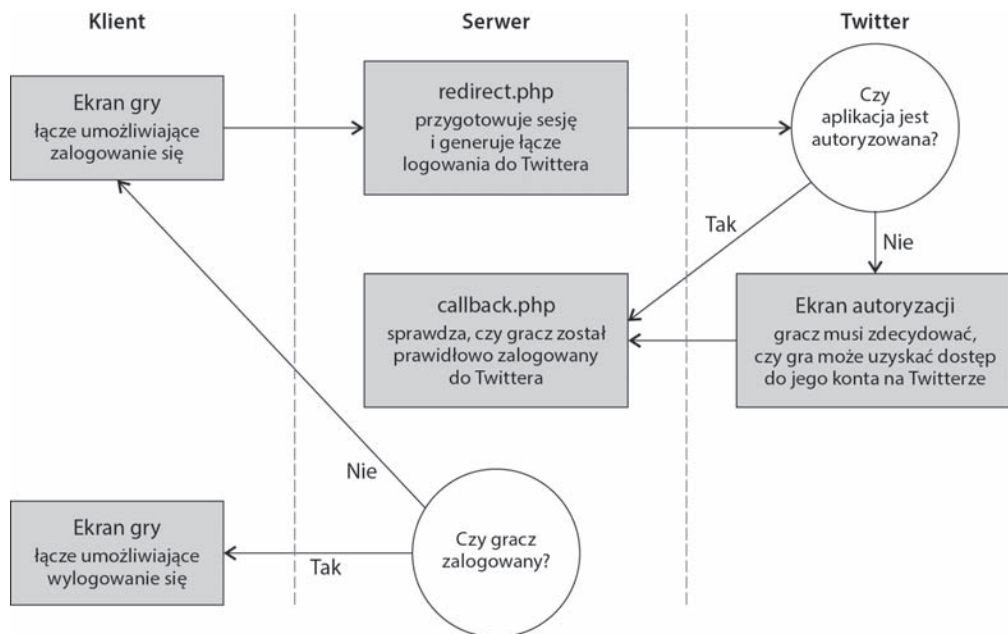
W miejscach (1) i (2) trzeba wpisać wartości wyświetlone na stronie informacji o utworzonej aplikacji Twittera. W miejscu (3) należy podać adres URL skryptu zwrotnego *callback.php*.

Ostatnim krokiem jest zmodyfikowanie skryptu *callback.php*, tak by zawierał prawidłowe przekierowanie do głównego pliku naszej gry:

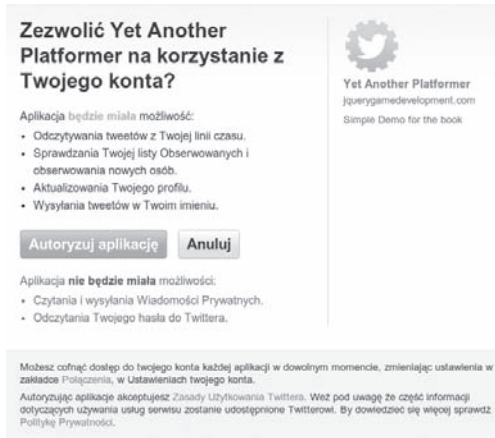
```
header('Location: ./index.php');
```

Uwierzytelnianie

Na poniższym diagramie został przedstawiony sposób przeprowadzania uwierzytelniania i autoryzacji użytkowników w grze za pośrednictwem Twittera:



W praktyce nie jest to tak skomplikowane, na jakie wygląda, przede wszystkim dlatego, że większość jest już zaimplementowana w bibliotece *twitteroauth*. Zajmiemy się utworzeniem strony logowania wyświetlanej po kliknięciu przycisku *Zaloguj przez Twittera*. Użyjemy zwykłego odsyłacza kierującego do pliku *redirect.php*. Kiedy użytkownik kliknie to łącze po raz pierwszy, zostanie przekierowany na stronę Twittera, na której będzie mógł autoryzować dostęp do konta:



Po wyrażeniu zgody nastąpi przekierowanie na adres, który został ustawiony w pliku *callback.php*. Jeśli użytkownik już raz wyraził zgodę na autoryzację, będzie mógł od razu się zalogować.

W skryptach JavaScript bardzo przyda nam się teraz informacja o tym, czy gracz jest połączony, czy nie. Aby ją uzyskać, musimy przekształcić główny dokument HTML gry w plik PHP i umieścić na jego początku następujący kod:

```
<?php
session_start();

require_once('twitter/twitteroauth/twitteroauth.php');
require_once('twitter/config.php');

/* Pobranie tokenów użytkownika z sesji. */
$access_token = $_SESSION['access_token'];
$connection = new TwitterOAuth(CONSUMER_KEY, CONSUMER_SECRET,
$access_token['oauth_token'], $access_token['oauth_token_secret']);
$user = $connection->get('account/verify_credentials');
?>
```

W tym kodzie rozpoczynamy sesję, dołączamy dwa pliki biblioteki twitteroauth oraz odczytujemy token sesji. Jeśli token jest ustawiony, oznacza to, że użytkownik jest zalogowany do Twittera.

Następnie serwer łączy się z Twitterem, aby pobrać obiekt użytkownika. Tak to wygląda po stronie serwera, ale kod JavaScript wciąż nie ma informacji o zalogowaniu. W związku z tym musimy utworzyć skrypt przesyłający dane do kodu klienckiego:

```
<script type="text/javascript">
<?php if($_SESSION['status'] == 'verified'){ ?>
    var twitter = true;
    var twitterName = "<?php print $user->screen_name; ?>";
<?php } else { ?>
    var twitter = false;
<?php } ?>
</script>
```

Jeśli gracz jest zalogowany do Twittera, globalna zmienna `twitter` ma wartość `true`, a zmienna `twitterName` zawiera nazwę użytkownika.

Ostatnim, o co należy zadbać, jest poinformowanie użytkownika o pomyślnym zalogowaniu za pomocą Twittera i umożliwienie wylogowania się. W tym celu musimy wprowadzić drobną zmianę w kodzie odpowiedzialnym za ekran początkowy wyświetlany po zalogowaniu:

```
<div id="startScreen" class="screen">
  <?php if($_SESSION['status'] != 'verified'){ ?>
    <a class="button tweetLink" href="./twitter/redirect.php">Zaloguj przez
      ↪Twittera</a>
  <?php } else { ?>
    <a class="button tweetLink" href="./twitter/clearsessions.php">Wyloguj z
      ↪Twittera</a>
  <?php }?>
  <a id="startButton" class="button" href="#">Rozpocznij grę</a>
</div>
```

Po wprowadzeniu powyższych modyfikacji można uznać, że uwierzytelnianie przez Twittera jest zaimplementowane.

Publikowanie najlepszych wyników na Twitterze

Teraz, kiedy gracz jest już zalogowany do Twittera, można publikować jego wyniki w znacznie wygodniejszy sposób niż poprzednio. Zaczniemy od utworzenia po stronie serwera skryptu `twitterPost.php`, w którym użyjemy interfejsu `statuses/update` API Twittera.

Spójrzmy na kompletny skrypt:

```
<?php
session_start();
require_once('twitter/twitteroauth/twitteroauth.php');
require_once('twitter/config.php');

$time = $_SESSION['time'];
$level = $_SESSION['level'];
if (isset($time) && isset($level)) {
    /* Pobranie tokenów użytkownika z sesji. */
    $access_token = $_SESSION['access_token'];
    $connection = new TwitterOAuth(CONSUMER_KEY, CONSUMER_SECRET,
    ↪$access_token['oauth_token'], $access_token['oauth_token_secret']);

    $parameters = array('status' => 'Właśnie udało mi się skończyć '.$level.'
    ↪poziom gry Yet Another Platformer w '.$time.' sekund!');
    $status = $connection->post('statuses/update', $parameters);
}
?>
```

Najprawdopodobniej rozpoznasz większość kodu z powyższego listingu, który dodaliśmy na początku skryptu ekranu początkowego (tylko wyróżniony kod jest nowy). W ostatnich dwóch wierszach tworzymy, a następnie przesyłamy do Twittera tekst, który chcemy opublikować, co

— jak widać — jest całkiem proste. Musimy jednak zająć się jeszcze jednym. Ponieważ gracz jest zalogowany, znamy jego nazwę, więc możemy ją wykorzystać na ekranie tablicy wyników.

W kodzie klienta utworzymy trochę inną od dotychczasowej wersję tablicy wyników:

```
$.ajax({
  dataType: "json",
  url: "highscore.php",
  data: {
    //...
  },
  async: false,
  success: function (json) {
    var top = "";
    for (var i = 0; i < json.top.length; i++) {
      if (json.intop && json.pos === i) {
        if (twitter) {
          top += "<input id='name' type='hidden' value='"+twitterName+"' />"
            + twitterName + " " + minSec(json.top[i].time)
            + " <a id='saveScore' href='#'>Zatwierdź</a>"
            + " <a id='tweetScore' href='#'>Tweetnij</a> <br>";
        } else {
          top += "<input id='name' placeholder='____' size='5' />"
            + " " + minSec(json.top[i].time)
            + " <a id='saveScore' href='#'>submit</a>"
            + " <a target='_blank' href='http://twitter.com/home?
↳status="+escape("właśnie udało mi się skończyć "+current
↳Level+" poziom gry Yet Another Platformer w "+minSec
↳(json.top[i].time)+" sekund!")+"'>Tweetnij</a> <br>";
        }
      } else {
        top += " " + json.top[i].name + " " + minSec(json.top[i].time) +
          ↳"<br>";
      }
    }
    $("#top_list").html(top);
  }
});
```

Tworzymy tu niewidoczne pole wejściowe przechowujące nazwę użytkownika pochodzącą z Twittera. Następnie wypisujemy tę nazwę na ekranie tablicy wyników. Zwróć uwagę, że nie musieliśmy w ogóle zmieniać kodu po stronie serwera.

Tyle na temat integracji z Twitterem. Jeżeli chcesz dowiedzieć się więcej i rozszerzyć funkcjonalność gry, zajrzyj do dokumentacji API Twittera.

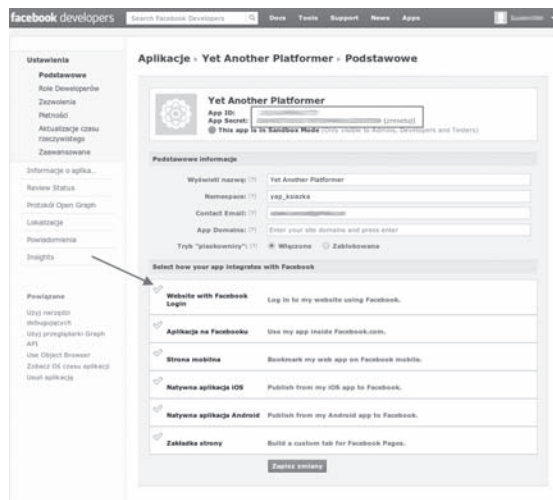
Integracja z Facebookiem

Integracja z Facebookiem i Twitterem jest w wielu aspektach podobna. Facebook jest jednak w dużo większym stopniu nastawiony na gry. Zajmiemy się implementacją osiągnięć dla zalogowanych użytkowników. Skorzystamy z SDK Facebooka dla PHP, ale są również dostępne rozwiązania dla innych języków.

Tak jak w przypadku Twittera, tu również musimy zacząć od zarejestrowania aplikacji w Facebooku. W tym celu trzeba się zalogować do serwisu dla deweloperów (<https://developers.facebook.com>) i kliknąć pozycję *Apps*¹.



Następnie należy kliknąć łącze *Create New App* i wypełnić obowiązkowe pola formularza. Po ich zatwierdzeniu pojawi się ekran z informacjami dotyczącymi nowej aplikacji. Podobnie jak było w przypadku Twittera, tu również najważniejsze są dwie wartości, co widać na rysunku:



¹ Aby móc tworzyć aplikacje, trzeba się zarejestrować jako programista. W tym celu musisz kliknąć przycisk *Register as a Developer* — *przyp. tłum.*

Na powyższym zrzucie strzałką została oznaczona sekcja umożliwiająca wybór sposobu integracji aplikacji z Facebookiem. Aby uzyskać pełny dostęp do interfejsu Open Graph API Facebooka, który umożliwia między innymi publikowanie osiągnięć, trzeba zaznaczyć pozycję *Aplikacja na Facebooku*.

Dzięki temu gra może być uruchamiana w ramce (iframe) wewnątrz strony Facebooka. Jednak aby to było możliwe, konieczny jest prawidłowy certyfikat dla HTTPS obowiązujący w domenie, w której jest dostępna gra. Jeśli jednak chcemy, by gra była ładowana z naszego serwera, nie ma tych wymagań, ale i tak trzeba wpisać adres w odpowiednim polu (aby adres został uznany za poprawny, należy podać `https://`, nawet jeśli nie ma się certyfikatu).

Jest jeszcze jedno, co należy zrobić, by aplikacja mogła publikować osiągnięcia — należy ją zarejestrować jako grę. W tym celu wystarczy kliknąć pozycję *Informacje o aplikacji*, a następnie w sekcji *Informacje o aplikacji* w polach *Kategoria* ustawić pozycję *Gry* i ustalić gatunek gry, co widać na poniższym rysunku:

The screenshot shows the 'Informacje o aplikacji' (App Information) page in the Facebook App Center. The breadcrumb trail is 'Aplikacje > Yet Another Platformer > Informacje o aplikacji'. On the left, there is a navigation menu with items like 'Ustawienia', 'Informacje o apl...', 'Review Status', 'Protokół Open Graph', 'Lokalizacja', 'Powiadomienia', 'Insights', and a 'Powiązane' section with links to developer tools. The main content area includes an 'Approval Status' section with an 'Unsubmitted' checkbox and a note that the app detail page has not been submitted for review. Below this are buttons for 'Web Preview', 'Submit App Detail Page', and 'Go to Review Status'. The 'Informacje o aplikacji' section contains several input fields: 'Język podstawowy' (set to 'Polski'), 'Wyświetl nazwę' (set to 'Yet Another Platformer'), 'Wiersz znacznika', 'Opis', 'Szczegółowy opis', 'Explanation for Permissions', and 'Publisher (optional)'. At the bottom, the 'Kategoria' (Category) is set to 'Gry' (Games) and the sub-category is 'Arkadowe' (Casual).

Uwierzytelnianie przez Facebooka

Podstawowy mechanizm uwierzytelniania jest bardzo podobny do tego z Twittera, ale istnieje pewna różnica w podejściu do dostępu. W przypadku Twittera można było zdecydować, czy aplikacja ma prawa tylko do odczytu, czy również zapisu. Facebook oferuje znacznie bogatsze możliwości ustalania poziomów dostępu, przy czym decyduje się o nich w czasie logowania.

Kod odpowiedzialny za uwierzytelnianie został przedstawiony na poniższym listingu. Tu również, jak w przypadku Twittera, musimy umieścić na początku skryptu kod pobierający obiekt bieżącego użytkownika.

```

<?php
session_start();

// Twitter...

// Facebook
require 'facebook/facebook.php';

$app_id = '(1)';
$app_secret = '(2)';
$app_namespace = 'yap_bookdemo';
$app_url = 'http://yetanotherplatformer.com/';
$scope = 'publish_actions';

$facebook = new Facebook(array(
    'appId' => $app_id,
    'secret' => $app_secret,
));

// Pobiera obiekt bieżącego użytkownika
$facebookUser = $facebook->getUser();
?>

```

Wyróżniony wiersz kodu odpowiada za wskazanie, że z poziomu gry będziemy chcieli publikować wpisy na osi czasu użytkownika. W miejscach (1) i (2) należy wpisać wartości wyświetlone na stronie konfiguracji aplikacji w Facebooku.

Jeśli zmienna `$facebookUser` to `null`, oznacza to, że użytkownik jest już zalogowany. W przeciwnym przypadku musimy wyświetlić przycisk logowania. Musimy więc napisać kod podobny do tego, który służył do wyświetlenia przycisku logowania za pośrednictwem Twittera:

```

<div id="startScreen" class="screen">
    ...
    <?php if (!$facebookUser) {
        $loginUrl = $facebook->getLoginUrl(array(
            'scope' => $scope,
            'redirect_uri' => $app_url
        ));
        ?>
        <a class="button tweetLink" href="<?php print $loginUrl; ?>">Zaloguj przez
        ↪ Facebooka</a>
    <?php } else {
        $logoutUrl = $facebook->getLogoutUrl(array(
            'next' => $app_url
        ));
        ?>
        <a class="button tweetLink" href="<?php print $logoutUrl; ?>">Wyloguj z
        ↪ Facebooka</a>
    <?php } ?>
    <a id="startButton" class="button" href="#">Rozpocznij grę</a>
</div>

```

Jak widać, Facebook SDK oferuje metodę generowania adresów URL służących do logowania i wylogowywania.

Do skryptu musimy jeszcze dopisać fragment służący do informowania javascriptowego kodu o tym, czy użytkownik jest zalogowany, czy nie. Użyjemy tu rozwiązania, które zastosowaliśmy w przypadku Twittera:

```
<script type="text/javascript">
  //...
  <?php if($facebookUser){ ?>
    var facebook = true;
    var facebookId = "<?php print $facebookUser; ?>";
  <?php } else { ?>
    var facebook = false;
  <?php } ?>
</script>
```

Tworzenie osiągnięć

Zajmiemy się teraz tworzeniem osiągnięć. Aby to było możliwe, na serwerze trzeba umieścić dwa pliki:

- plik HTML zawierający w nagłówku zestaw znaczników meta,
- plik obrazka reprezentującego osiągnięcie na osi czasu gracza.

Dokument HTML będzie nie tylko plikiem konfiguracyjnym, ale będzie również do niego prowadził odsyłacz z wpisu z osiągnięciem na osi czasu gracza. Aby Facebook go rozpoznał, trzeba w jego nagłówku umieścić siedem znaczników meta:

- `og:type` — zawierający wartość `game.achievement`. Odróżnia on osiągnięcie od innych wpisów OpenGraph.
- `og:title` — zawierający krótki opis osiągnięcia.
- `og:url` — zawierający adres URL bieżącego pliku.
- `og:description` — zawierający dłuższy opis osiągnięcia.
- `og:image` — zawierający adres pliku obrazka, o którym była mowa wcześniej. Może to być plik PNG, JPEG lub GIF o minimalnych wymiarach 50×50 pikseli i maksymalnym stosunku wymiarów 3:1.
- `game:points` — zawierający liczbę punktów powiązaną z danym osiągnięciem. Wartość ta nie może przekroczyć 1000 punktów i być mniejsza niż 1. Osiągnięcia o wyższej wartości będą z większym prawdopodobieństwem wyświetlane u znajomych gracza.
- `fb:app_id` — będący identyfikatorem aplikacji.

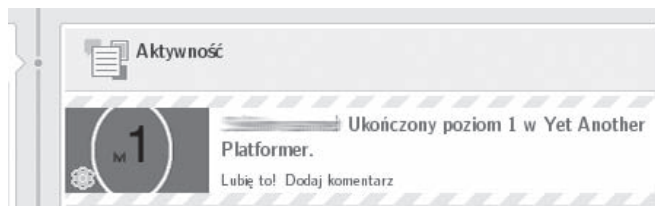
W ciele dokumentu HTML można umieścić dokładny opis osiągnięcia oraz inne informacje. Prosty przykład kompletnej strony osiągnięcia znajduje się na poniższym listingu:

```

<html>
  <head>
    <meta property="og:type" content="game.achievement" />
    <meta property="og:title" content="Ukończony poziom 1" />
    <meta property="og:url" content="http://8bitentropy.com/yap/ach1.html" />
    <meta property="og:description" content="Właśnie zakończyłeś pierwszy
    ↳ poziom!" />
    <meta property="og:image" content="http://8bitentropy.com/yap/ach1.png" />
    <meta property="game:points" content="50" />
    <meta property="fb:app_id" content="(1)" />
  </head>
  <body>
    <h1>Świetna robota, pierwszy poziom już za tobą!</h1>
  </body>
</html>

```

Osiągnięcie zostanie wyświetlone w sposób zbliżony do tego, co widać na poniższym rysunku:



Jednak samo przygotowanie dokumentu nie wystarczy — trzeba go opublikować na Facebooku. Aby to zrobić, musimy wykonać żądanie POST pod określony adres i przekazać niezbędne parametry. Żądanie to musi być również powiązane z tokenem aplikacji. Za pomocą tokenów Facebook upewnia się, że komunikuje się z nim określona aplikacja, a nie jakaś inna.

Musimy więc napisać skrypt przesyłający osiągnięcia do Facebooka. Kompletny kod został przedstawiony na poniższym listingu:

```

<?php
require 'facebook/facebook.php';

$app_id = '(1)';
$app_secret = '(2)';
$app_namespace = 'yap_bookdemo';
$app_url = 'http://yetanotherplatformer.com/';
$scope = 'publish_actions';

$facebook = new Facebook(array(
    'appId' => $app_id,
    'secret' => $app_secret,
));

```



```

$app_access_token = get_app_access_token($app_id, $app_secret);
$facebook->setAccessToken($app_access_token);

$response = $facebook->api('/(1)/achievements', 'post', array(
    'achievement' => 'http://yetanotherplatformer.com/ach1.html',
));

print($response);

// Funkcja pomocnicza pobierająca token dostępu aplikacji (access token)
function get_app_access_token($app_id, $app_secret) {
    $token_url = 'https://graph.facebook.com/oauth/access_token?'
        . 'client_id=' . $app_id
        . '&client_secret=' . $app_secret
        . '&grant_type=client_credentials';

    $token_response = file_get_contents($token_url);
    $params = null;
    parse_str($token_response, $params);
    return $params['access_token'];
}
?>

```

Przeznaczenie poszczególnych fragmentów kodu jest dosyć jasne, a poza tym większość pojawiła się już wcześniej. W wyróżnionym fragmencie pobieramy token aplikacji, wiążemy go z tworzonym żądaniem, a następnie — za pomocą SDK — wysyłamy żądanie POST.

Adres żądania ma następującą postać: *identyfikator aplikacji/osiągnięcie*. Przekazywany parametr jest zwykłym adresem URL pliku osiągnięcia.

Ponieważ generowany komunikat o błędzie (jeśli coś pójdzie nie tak) może być niejasny, poszukiwania przyczyn problemów najlepiej zacząć od sprawdzenia poprawności pliku osiągnięcia za pomocą narzędzia udostępnianego przez Facebook (<https://developers.facebook.com/tools/debug/>).

Publikowanie osiągnięć

Po zarejestrowaniu osiągnięcia w Facebooku możemy je przyznać graczom. Polecenie to jest również przesyłane w formie żądania POST, które musi być powiązane z tokenem aplikacji. Aby uprościć przykład, utworzymy niewielki skrypt PHP, który — po wywołaniu — będzie nadawał użytkownikom osiągnięcia. Takiego rozwiązania nie powinno się jednak stosować w rzeczywistych aplikacjach, ponieważ zwykle chcemy uniknąć sytuacji, w której użytkownik sam musi uruchomić jakiś skrypt. W związku z tym najlepszym wyjściem byłoby umieszczenie omawianego kodu w pliku *highscore.php*.

Poniżej znajduje się pełny kod skryptu. Jest on bardzo podobny do tego, który odpowiadał za zarejestrowanie osiągnięć; istotne różnice zostały wyróżnione.

```

<?php
session_start();

// Facebook
require 'facebook/facebook.php';

$app_id = '(1)';
$app_secret = '(2)';
$app_namespace = 'yap_bookdemo';
$app_url = 'http://yetanotherplatformer.com/';
$scope = 'publish_actions';

$facebook = new Facebook(array(
    'appId' => $app_id,
    'secret' => $app_secret,
));

// Pobiera obiekt bieżącego użytkownika
$facebookUser = $facebook->getUser();

$app_access_token = get_app_access_token($app_id, $app_secret);
$facebook->setAccessToken($app_access_token);

$response = $facebook->api('/'.$facebookUser.'/achievements', 'post',
    array(
        'achievement' => 'http://yetanotherplatformer.com/ach1.html'
    ));

print($response);

// Funkcja pomocnicza pobierająca token dostępu aplikacji (access token)
function get_app_access_token($app_id, $app_secret) {
    ...
}

?>

```

Tym razem tworzymy żądanie POST pod adres w formacie *identyfikator użytkownika/osiągnięcia*. Teraz w kodzie gry, po ukończeniu pierwszego poziomu, wystarczy wykonać asynchroniczne wywołanie tego pliku:

```

if (status == "finished") {
    ...
    if (facebook && currentLevel === 1) {
        $.get("ach1.php");
    }
    ...
}

```

Podsumowanie

W tym rozdziale omówiliśmy naprawdę sporo zagadnień, przy czym niektóre, jak choćby różne możliwości integracji z serwisami społecznościowymi, jedynie zasygnalizowaliśmy. Interfejsy API Facebooka i Twittera są bardzo rozbudowane i wciąż się zmieniają. Jeśli chcesz z nich skorzystać w najlepszy możliwy sposób, musisz na bieżąco śledzić ich dokumentację.

Trzeba jednak pamiętać, że korzystanie z zewnętrznych serwisów, zwłaszcza bezpłatnych, powoduje powstanie silnych zależności, co może mieć poważne konsekwencje. Serwisy te mogą w dowolnej chwili zmienić API, wymuszając w ten sposób konieczność szybkiego wprowadzenia zmian w kodzie aplikacji. Może się też okazać, że opublikowana gra nie będzie od jakiegoś momentu obsługiwana. W związku z tym zawsze trzeba mieć przygotowane alternatywne rozwiązanie.

W kolejnym rozdziale zajmiemy się innym gorącym tematem — przystosujemy grę do działania na urządzeniach mobilnych. Skorzystamy z opracowanej wcześniej gry MMORPG i zmodyfikujemy ją tak, by działała na smartfonach i tabletach.

Skorowidz

A

adres URL, 31, 156, 167
akcja, 16
Alien Breed, 87
Android, 173, 174
animacja, 29
 dezaktywowanie, 32
 gracza, 76
 implementowanie, 30
 obiekt, *Patrz:* obiekt animacji
 prędkość odtwarzania, 30, 31
 przeciwnika, 76
 tła, 76
 w pętli, 48
 zatrzymanie, 18
aplikacja
 ikona, 190
 nasłuchująca, 153
 natywana, 173
 token, 168, 169
 webowa, 173
 wymagane pliki, 192
Aptana, 12
arkusz, 29
 sprite'ów, *Patrz:* sprite
atrybut
 data, 21
 src, 29

B

biblioteka, 28
 jQuery, *Patrz:* jQuery
 jQuery Mobile, *Patrz:* jQuery Mobile
 odtworzenia dźwięku, 196

OpenAL, 204
SFXR.js, 214
SoundJS, 213
SoundManager, 211, 212, 213
 twitteroauth, *Patrz:* twitteroauth
BrowserQuest, 87, 98

C

Chrome, 195, 200, 204
Chrono Trigger, 87
Civilization, 87
CreateJS, 213
CSS przekształcenie, 57, 64, 65
 obrót, 65
 skalowanie, 65, 66
 trójwymiarowe, 65
CSS Transforms, *Patrz:* CSS przekształcenie
czas
 odmierzenie, 48
 zwłoki, 53

D

danych składowanie lokalne, 193
debugowanie, 27, 48
diagram przejść, 39
directional pad, *Patrz:* D-pad
dokumentacja API, 17, 23, 31
DOM, 21, 27
 modyfikowanie, 56, 61
 węzeł
 dodawanie, 21, 22
 przywracanie, 23
 usuwanie, 21, 23
duszek, *Patrz:* sprite

dziedziczenie, 83, 84
 dźwięk, 195, 198
 filtr, 211
 głośność, 207
 kilka źródeł, 208
 kompresor, 210
 konwolucja, 210
 opóźnienie, 209
 osadzanie, 195, 198
 sterowanie, 204
 w przestrzeni trójwymiarowej, 209

E

EasyPHP, 12
 Eclipse, 12
 edytor, 12
 Aptana, *Patrz:* Aptana
 Eclipse, *Patrz:* Eclipse
 Emacs, *Patrz:* Emacs
 Komodo Edit, *Patrz:* Komodo Edit
 map kafelków, *Patrz:* mapa kafelków edytor
 Notepad++, *Patrz:* Notepad++
 VIM, *Patrz:* VIM
 efekt
 dźwiękowy, 195, 204, 214
 paralaksy, 82
 poświaty, 190
 ekran
 domowy, 188, 189
 dotykowy, 180
 logowania, 125
 tytułowy, 190
 element
 audio, 195, 200, 213, 214
 canvas, 27, 178
 części wspólne, 71, 72, 73, 74
 div, 21, 29
 odłączanie, 61
 pozycjonowanie, 33
 DOM, 21, 27
 odłączanie, 61
 usuwanie, 21
 embed, 198
 fizyka ruchu, 78
 img, 29
 klon, 56
 nadrzędny, 20
 poruszający się, *Patrz:* sprite

renderowanie, 27, *Patrz:* renderowanie
 video, 200, 213, 214
 Emacs, 12
 eskejpowanie łańcuchów tekstowych, 127

F

Facebook, 143, 157
 dla deweloperów, 164, 165
 SDK, 164, 167
 Firefox, 195, 200, 204
 flaga, 64
 Flash, 25, 196, 211, 213
 format
 AAC, 200
 GIF, 178
 animowany, 29
 JPEG, 178
 JSON, 98, 111
 MIDI, 199
 MP3, 200
 Ogg Vorbis, 200
 PNG, 178
 TIFF, 178
 WAV, 199, 200
 formularz, 19
 framework, 27, 28
 Frogger, 26
 funkcja
 \$.ajax, 98, 99, 108, 112, 113
 alias, 112, 113
 debugowanie, 116
 skrót, 113
 \$.data, 64
 \$.get, 113
 \$.getJSON, 99, 113, 126, 127, 131
 \$.getScript, 113, 115
 \$.load, 113
 \$.post, 113
 .always, 116
 .animate, 17
 .append, 22
 .bind, 19, 21
 .clearQueue, 18
 .click, 19
 .complete, 116
 .css, 16, 30
 .data, 21
 .delay, 19
 .delegate, 20, 21

.dequeue, 19
 .detach, 23
 .done, 116
 .error, 116
 .fail, 116
 .find, 90
 .html, 22, 101
 .off, 21
 .on, 21
 .prepend, 22
 .remove, 23
 .stop, 18
 .success, 116
 .unbind, 20
 .undelegate, 20
 addCallback, 48
 animacyjna, 17
 data, 57
 fadeIn, 18
 fadeOut, 18
 fadeTo, 18
 get_browser, 177
 gf.importTiled, 98
 hash, 128
 haszująca, 128
 hide, 18
 json_encode, 129
 łańcuch, 22
 mysqli_escape, 127
 obsługi zdarzenia, 41, 182, 186
 niskopoziomowa, 19
 requestAnimationFrame, 58
 rgba, 101
 setInterval, 30, 48, 58
 setTimeout, 48, 58
 show, 18
 slideDown, 18
 slideUp, 18
 wykrywania
 położenia, 173, 174
 ruchu, 173, 174
 zwrotna, 48, 50, 52, 58, 116, 135

G

garbage collector, *Patrz:* odśmiecaacz
 Gauntlet, 87
 getter, 16
 GIF, *Patrz:* format GIF
 Gimp, 12

Google, 157
 gra
 dwuwymiarowa, 25, 61
 fabularna, *Patrz:* RPG
 hack and slash, 87
 implementowanie po stronie serwera, 138, 149
 integracja
 z ekranem domowym, 174
 z Facebookiem, 164, 165, 168
 z Twitterem, 157, 158, 159, 160, 161, 162
 izometryczna, 104
 kod, 28
 inicjalizujący, 37
 konsolowa, 181
 liczba graczy, 123
 logika, *Patrz:* logika
 MMORPG, *Patrz:* MMORPG
 mobilna, 173
 multiplayer, 123
 pełnoekranowa, 188
 pętla główna, 39, 40
 platformowa, 26, 55, 61, 108
 poziom, *Patrz:* poziom
 scena, *Patrz:* scena
 symulacyjna, 87
 testowanie, 175
 trójwymiarowa, 25
 wielopoziomowa, 117
 wojenna, 87
 gracz
 aktualizacja położenia, 134, 135
 dodawanie, 135
 hasło, 124
 konto, 124, 125, 128
 liczba żyć, 38
 logowanie, 124, 125, 131
 za pomocą konta na Facebooku, 165
 za pomocą konta na Twitterze, 143, 156, 157,
 160, 161
 osiągnięcie, 143
 pasek doświadczenia, 103
 pasek energii, 103
 rozmowa z NPC, 101
 siła ciosu, 102, 103
 synchronizacja, 132, 133
 usuwanie, 135
 walka, 102
 wznowienie rozgrywki, 130
 zabicie przeciwnika, 137
 grupa, 63

H

hitzone, *Patrz:* strefa uderzenia
HTML5, 193, 195, 200

I

instrukcja switch, 40, 80
interfejs
 API, 174, 200
 Audio Data, *Patrz:* Audio Data
 dotykowy, 180
 Open Graph API, 165
 użytkownika, *Patrz:* UI
 Web Audio API, *Patrz:* Web Audio API
 wielodotkowy, 173, 180
Internet Explorer, 27, 176, 200, 204
interwał, 48
 bazowy, 48
 normalizacja, 52
 uchwyty, *Patrz:* uchwyt interwałów
iPad, 176
iPhone, 176
iPod, 176
iPod Touch, 176

J

JavaScript, 48
język
 JavaScript, *Patrz:* JavaScript
 jednowątkowy, 48
 PHP, 144, 159
 stosowany po stronie serwera, 13
joystick, 181, 184
jQuery, 15, 213
 obiekt, 62
 samouczek, 23
jQuery Mobile, 174

K

kafelek, 61, 68, 69
 dodawanie, 93
 izometryczny, 104
 klasa, 69
 kolizje, 71, 72, 73, 74, 89
 semantyczny, 109
 widoczny, 89

kamera, 94, 95
klasa, 20, 64
 anonimowa, 77
klawiatura, 19
 odpytywanie, *Patrz:* odpytywanie klawiatury
klawisz
 powtarzanie, 53
 wciśnięty, 53
 wirtualny, 181
 zwolniony, 53
klient użytkownika, *Patrz:* UA
klucz, 21
Koch Peter-Paul, 176
kod
 HTML, 21
 obiekty, 77, 83
 optymalizacja, 47, 55
 zaciemnianie, 151, 152
 zorientowany obiektowo, 61
kodowanie wartości, 153
kolejka
 animacji, 18
 fx, 19
kolejkowanie, 18
kolizja, 96
 algorytm wykrywania, 61, 71, 72, 73, 74, 79,
 96, 110
 części wspólne, 71, 72, 73, 74
 postaci z otoczeniem, 97, 99, 101
 wykrywanie, 42
konsola, 25, 101, 103
kontekst, 204
kontroler D-pad, *Patrz:* D-pad
konwolucja, 210
korektor barwy, 204

L

liczba żyć, 21
literal obiektowy, 17
logika, 27, 39, 61, 110

Ł

łańcuch
 jako argument, 22
 parsowanie, 22
 pusty, 22
 tekstowy eskejpowanie, 127
 wywołań, 16, 17

M

macierz rzadka, 88
 MAMP, 12
 manifest, 192
 mapa
 kafelków, 61, 68, 69, 87
 dodatkowa, 108
 edytor, 98
 izometrycznych, 104
 optymalizowanie, 88
 pobieranie, 108
 przesuwanie, 90
 tworzenie, 69
 zaimportownie danych, 114
 klucz-wartość, 17
 Massively Multiplayer Online Role-Playing Game, *Patrz:* MMORPG
 maszyna stanów skończonych, 39, 40
 mechanizm
 lokalnego składowania danych, 193
 OAuth, *Patrz:* OAuth
 metoda
 \$.extend, 31, 49
 canPlayType, 201
 getItem, 193
 noteOff, 205
 noteOn, 205
 setItem, 193
 start, 205
 stop, 205
 miecz, 100
 MMORPG, 108, 123, 144, 173
 moduł mysli, 127
 modyfikator ataku, 102
 MySQL, 144
 myszy przycisk, 19

N

Non-Player Character, *Patrz:* NPC
 Notepad++, 12
 NPC, 95, 99, 101, 111

O

OAuth, 157
 obfuscating, *Patrz:* kod zaciemnianie

obiekt
 animacji, 31
 AudioParams, 207
 BufferLoader, 208
 Image, 36, 200
 jQuery, 62
 localStorage, 193
 sessionStorage, 193
 sound, 197
 obraz, 178
 obsługa zdarzenia, *Patrz:* zdarzenie obsługa
 obszar półprzezroczysty, 101
 odpytywanie
 klawiatury, 53, 55
 stanu, 53
 odsyłacz, 20
 odśmiecacz, 21
 odtwarzacz CD, 204
 okna przeglądarki skalowanie, 19
 Opera, 195, 200, 204
 operacja
 moduło, 30, 42
 przesunięcia w lewo, 154
 przesunięcia w prawo, 154
 oś czasu, 143

P

Pagella Andres, 103
 pamięć, 178
 parallax scrolling, *Patrz:* efekt paralaksy
 pasek
 stanu, 189
 doświadczenia, 103
 energii, 103
 perspektywa
 izometryczna, 103, 104
 mapy, 87
 Pixon, 12, 28
 plik
 callback.php, 161
 config-sample.php, 160
 dbconnect.php, 130
 dźwiękowy, 195, 198, 204
 ładowanie, 213
 ładowanie wstępne, 201, 203
 JSON, *Patrz też:* format JSON
 interpreter, 111
 konwersja, 114
 ładowanie, 113
 tworzenie, 126

- plik
 - ładowanie, 108, 113, 201, 203, 213
 - asynchroniczne, 113
 - łączenie, 107
 - manifest, 192
 - MIDI, 195
 - muzyczny, 199, 200
 - php_browscap.ini, 177
 - redirect.php, 160
 - wideo, 213
 - wymagany przez aplikację, 192
 - podpoziom, 108
 - polimorfizm, 16
 - postać niezależna, *Patrz:* NPC
 - potomek, 22
 - pośredni, 90
 - bezpośredni, 90
 - dodawanie, 22
 - poziom, 107
 - pozycjonowanie bezwzględne, 33
 - procesora prędkość, 178
 - projekcja ortogonalna, 87
 - przeglądarka, 25
 - desktopowa, 175
 - mobilna, 175, 176
 - szerokość strony, 179
 - tryb offline, 173, 174, 192
 - wykrywanie, 177
 - po stronie klienta, 175
 - po stronie serwera, 177
 - przekształcenie CSS, *Patrz:* CSS przekształcenie
 - przestrzeń nazw, 28, 64
 - przezroczystość, 101
- ## R
- referencja cykliczna, 21
 - renderowanie, 27
 - RPG, 26, 55, 87, 108
 - rzutu kostką symulacja, 102
- ## S
- Safari, 176, 195, 200, 204
 - scena, 27
 - selektor, 16, 90
 - serwer, 12
 - EasyPHP, 99, *Patrz:* EasyPHP
 - MAMP, 99, *Patrz:* MAMP
 - WWW, 99
 - XAMPP, *Patrz:* XAMPP
 - sesja, 130
 - aktywna, 130
 - token, 161
 - setter, 17
 - sieć społecznościowa, 143
 - siła ciosu, 102, 103
 - SimCity, 87
 - skalowalność, 123
 - skrypt
 - JavaScript, 126
 - konfiguracyjny twitteroauth, 160
 - mediaelement.js, 214
 - tablicy wyników, *Patrz:* tablica wyników skrypt
 - tworzenie przeciwników, 112
 - zasięg globalny, 115, 120
 - zewnętrzny, 115
 - Sonic the Hedgehog, 61
 - sprite, 20, 27, 28, 38
 - animowany, 27, 29
 - arkusz, 29
 - ładowanie, 111
 - nazwa, 62
 - obszar kolizji, 99
 - przekształcenie, 64, 65
 - przemieszczanie, 33
 - przesłanianie, 95, 96
 - redukowanie liczby, 179
 - tworzenie, 55
 - SQL wstrzyknięcie, 127
 - SQL injection, *Patrz:* SQL wstrzyknięcie
 - state polling, *Patrz:* odpytywanie stanu
 - strefa uderzenia, 100
 - struktura DOM, *Patrz:* DOM
 - Super Mario Bros, 61
 - system
 - Android, *Patrz:* Android
 - iOS, 173, 174, 176
- ## T
- tabela graczy, 124
 - tablica wyników, 143
 - oszukiwanie, 149, 150, 153, 154, 155
 - publikowanie, 162, 167, 168, 169
 - skrypt, 143
 - tworzenie, 144
 - weryfikacja, 149
 - wyświetlanie, 147

technologia
 MySQL, 123, *Patrz:* MySQL
 PHP, 123, *Patrz:* język PHP
 tekstu wyświetlanie, 101
 The Legend of Zelda: A Link to the Past, 87
 Tiled, 98
 tło, 27, 29
 kolor, 101
 położenie, 29
 top-down perspective, *Patrz:* perspektywa mapy
 transformacja
 anizotropowa, 66
 izotropowa, 66
 tweet
 publikowanie, 156
 tekst, 156
 Twitter, 143, 156, 157
 dla deweloperów, 158, 159
 twitteroauth, 159, 160
 skrypt konfiguracyjny, 160

 DynamicsCompressorNode, 210
 PannerNode, 209
 ScriptProcessorNode, 209
 tworzenie, 207
 typ, 209
 WaveShaperNode, 211
 wzmocnienia, 207
 źródłowy, 205
 widok izometryczny, 88
 właściwość, 16, 17
 background-position, 40
 CSS filter, 64
 navigator.userAgent, 175
 readyState, 201
 transform, 65
 z-index, 95, 101, 104
 zmiana w czasie, 17
 wydajność, 21, 27, 48, 177
 ograniczenia, 178, 179
 wywołanie międzydomenowe, 113
 wzmacniacz, 204, 207

U

UA, 175, 177
 spoofing, 176
 uchwyt interwałów, 32
 UglifyJS, 152
 UI, 27, 144
 urządzenie mobilne, 173
 orientacja, 191
 wydajność, 173, 174
 User Agent, *Patrz:* UA

V

VIM, 12
 Vleugels Kenney, 75

W

Warcraft, 87
 wartość, 21
 wątek roboczy, 48
 Web Audio API, 195, 204, 208, 213
 węzeł, 204
 AnalyserNode, 210
 BiquadFilterNode, 211
 ConvolverNode, 210
 DelayNode, 209

X

XAMPP, 12

Z

zapytanie SQL, 127
 zasobu pobieranie, 34
 zdarzenie
 deviceorientation, 191
 keydown, 41, 53
 keyup, 53
 obsługa, 19, 20, 182, 183, 186
 touchend, 183
 touchstart, 182
 wielodotyku, 182
 Zelda, 87
 zmienna
 dodatkowa, 155
 nazwa, 154
 znacznik
 embed, 198
 img, 200
 link, 190
 meta
 apple-mobile-web-app-capable, 188
 apple-mobile-web-app-status-bar-style, 189

znak

%, 30

<<, 154

>>, 154

ż

żądanie, 144

GET, 127

POST, 127, 168, 169

synchroniczne, 99, 115

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>

jQuery

Niezbędnik programisty gier

JavaScript jeszcze nigdy w historii nie był tak popularny. Jego możliwości w połączeniu ze współczesnymi przeglądarkami są oszałamiające. Nikogo nie dziwią już aplikacje, które swoją użytecznością przewyższają tradycyjne desktopowe odpowiedniki. Teraz przyszła kolej na gry. Czy wkrótce i one podbiją rynek?

Dzięki bibliotece jQuery korzystanie z potencjału JavaScriptu stało się zdecydowanie łatwiejsze. To sprawiło, że zyskała ona ogromną popularność i jest ceniona w środowisku programistów. W trakcie lektury tej książki odkryjesz, jak dzięki jQuery sprawnie stworzyć wciągającą grę.

Naucz się tworzyć gry oparte na sprite'ach, wspierające tryb multiplayer oraz zintegrowane z sieciami społecznościowymi. Dowiedz się, jak wykrywać kolizje, tworzyć rzuty izometryczne oraz projektować gry mobilne. Już za chwilę będziesz w stanie stworzyć swoją własną platformówkę, a może nawet prostą grę RPG. Sięgnij po tę książkę i przekonaj się, że to nie takie trudne!

Masz pomysł na grę? Zrealizuj go z jQuery!



Dzięki tej książce:

- poznasz zaawansowane możliwości języka JavaScript i biblioteki jQuery
- nauczysz się opierać animację na sprite'ach
- poradzisz sobie z ograniczeniami urządzeń mobilnych
- stworzysz wciągającą grę

helion.pl
księgarnia
internetowa

Nr katalogowy: 18258



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>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-246-8608-7



Cena: 39,90 zł

Informatyka w najlepszym wydaniu

9 788324 686087