1

Quiz

Jest to rodzaj gry o bardzo prostych zasadach. Trzeba udzielić odpowiedzi na pytania i albo się je zna, albo się zgaduje. Jest to niezwykle popularny typ gry, można go spotkać m.in. w interaktywnych teleturniejach telewizyjnych. Nawet jeśli gra jest bardziej skomplikowana niż prosty zestaw pytań i odpowiedzi, to tak jak każdy program, działa w oparciu o pewną logikę. Gdy król spyta Cię, czy chcesz walczyć ze smokiem, i odpowiesz „tak", to jest to mały prosty quiz. Przesadą byłoby stwierdzenie, że wpadnięcie w otchłań w platformówce albo strata wszystkich punktów mocy w grze RPG jest tym samym, czym udzielenie odpowiedzi w quizie, ale jednak programowanie zasad gry i ich konsekwencji w każdym gatunku wygląda podobnie.

Receptura. Tworzenie zbioru pytań

Biorąc pod uwagę fakt, że tę książkę mogą czytać początkujący, starałem się, aby objaśnienia w tym rozdziale były jak najprostsze. Dalsze rozdziały są bardziej skomplikowane. Natomiast materiał znajdujący się w tym rozdziale ma umożliwić zrozumienie tematu każdemu. Każdy od czegoś zaczyna i dla niektórych takim początkiem może być właśnie ten tekst. Jeśli treść tego rozdziału wydaje Ci się banalna, możesz go tylko przejrzeć albo w ogóle pominąć. W dalszych rozdziałach są opisane bardziej złożone i trudniejsze zagadnienia.

Opis gry w tym rozdziale pełni trzy funkcje. Po pierwsze, chcę na jego bazie objaśnić podstawy języków HTML, CSS i JavaScript. Z tych trzech technologii najważniejszy jest język JavaScript. Jeśli słabo znasz podstawy tego języka, przeczytaj poradnik znajdujący się w dodatku A. Po drugie, w opisywanych grach jest używanych wiele różnych bibliotek i chcę mieć pewność, że będziesz wiedzieć, skąd je brać. Po trzecie, chcę ustalić wygodne i powtarzalne zasady tworzenia, edytowania, zapisywania i otwierania plików będących podstawą tej książki.

Jeśli nie masz edytora tekstu, to musisz się w niego zaopatrzyć. Edytorów do tworzenia i edytowania plików HTML, CSS oraz JavaScript jest mnóstwo. Jeśli nie wiesz, który wybrać, możesz skorzystać z podpowiedzi zawartych w dodatku C.

Jeśli masz już edytor tekstu, uruchom go, utwórz w nim plik *quiz/start/index.html* i wklej do niego kod z listingu 1.1. Jeśli jeszcze nie pobrałeś plików z serwera FTP, poszukaj informacji o nich we wstępie.

Listing 1.1. Plik index.html — struktura dokumentu HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Quiz</title>
    <link rel="stylesheet" type="text/css" href="main.css">
  </head>
  <body>
    <h1>Quiz</h1>
    <div id="quiz">
    </div>
  </body>
</html>
```

UWAGA

Akronim HTML pochodzi od angielskich słów *Hypertext Markup Language* (język znakowania hipertekstu). Nazwa ta powstała w dawnych czasach, gdy **łącza** nazywano także **hiperłączami** i istniały jeszcze inne technologie pozwalające na przemieszczanie się między dokumentami. Hipertekst to zwykły tekst zawierający łącza. Znaczniki to specjalne fragmenty tekstu otaczające zwykły tekst w celu określenia jego funkcji. Krótko mówiąc, HTML to zestaw wytycznych składniowych określających sposób łączenia różnych rodzajów tekstu w celu uzyskania wzajemnie powiązanych stron będących plikami z rozszerzeniem *.html*.

Znacznik HTML to tekst znajdujący się w <nawiasie trójkątnym>. Natomiast **element** HTML to para znaczników, <otwierający> i <zamykający>, wraz z tym, co znajduje się między nimi (zwróć uwagę na znak / w znaczniku zamykającym).

Tworzenie dokumentu zaczyna się od zadeklarowania jego typu przy użyciu deklaracji `DOCTYPE`. Dzięki temu przeglądarka internetowa „wie", że otrzymała do przetworzenia dokument HTML. Przeglądarki mogą też otwierać dokumenty innych typów, od plików XML przez pliki dźwiękowe po obrazy graficzne. Dlatego też zadeklarowanie typu dokumentu jako normalnej strony internetowej jest bardzo ważne. Pewnie zastanawia Cię, co się stanie, jeśli tego nie zrobisz. W zależności od przeglądarki skutki mogą być zarówno mało znaczące, jak i straszne. Bądź co bądź tak naprawdę nigdy nie wiadomo, co się stanie, i dlatego najlepiej jest nie zapominać o tym drobiazgu na początku dokumentu.

Na drugim miejscu znajduje się znacznik `<html>`. Jest to globalny kontener dokumentu, który zazwyczaj zawiera dwa elementy: `head` i `body`, jak widać na powyższym listingu. Zwróć uwagę, że wszystkie wymienione elementy składają sie ze znacznika otwierającego i zamykającego z ukośnikiem (`/`). Znacznik zamykający pozwala umieścić w elemencie inny element.

W elemencie `head`, ogólnie rzecz biorąc, wpisuje się informacje, które są ważne dla przeglądarki internetowej, ale nie mają bezpośredniego wpływu na to, co użytkownik widzi na ekranie. Element `meta` ma wiele zastosowań. W tym przypadku został użyty do określenia sposobu kodowania dokumentu. Jeśli nie określi się kodowania, to znaki niewchodzące w skład standardowego (i mocno ograniczonego) zestawu będą traktowane w nieprzewidywalny sposób. Jeśli będziesz pisać np. tylko po angielsku, to pewnie nie napotkasz trudności, ale z typowo polskimi literami typu ś, ć możesz mieć problemy. Nawet w konsoli JavaScript (a także narzędziu Firebug i narzędziach dla webmasterów przeglądarki Chrome) jest wyświetlane powiadomienie, że brakuje tego składnika. Mimo to w tej książce w większości przypadków ten element jest opuszczony, aby można było skupić się na tym, co jest nowego w każdym rozdziale.

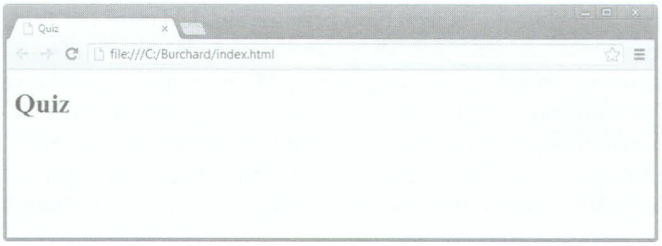
Treść elementu `title` jest wyświetlana w górnej części okna przeglądarki, na górnej belce, karcie albo na obu. Wykorzystują ją też aplikacje do tworzenia skrótów i zakładek jako „zwięzły opis strony".

Dalej znajduje się element `link` z atrybutem `rel` ustawionym na `stylesheet`, atrybutem `type` informującym, że dołączany jest arkusz stylów CSS, oraz atrybutem `href` zawierającym ścieżkę do tego arkusza. W tym przypadku ścieżką jest nazwa pliku, co oznacza, że ten plik znajduje się w tym samym folderze co plik *index.html*. Element ten jest często używany do dołączania do stron arkuszy stylów i oprócz ścieżki praktycznie zawsze wygląda tak samo. W przypadku elementu `link`, jak również `meta` należy zwrócić uwagę na brak znacznika zamykającego (np. `</link>`). Elementy te nie są kontenerami i nie muszą mieć zamknięcia.

W elemencie `body` znajdują się dwa zagnieżdżone elementy. Pierwszy z nich to nagłówek `h1`, którego zawartość domyślnie jest wyświetlana powiększoną czcionką. Drugi to element `div` pełniący rolę kontenera oznakowanych informacji. W przedstawionym przykładzie element `div` ma atrybut `id`, który obok atrybutu `class` i samej nazwy elementu stanowi jeden z trzech najczęściej używanych sposobów wybierania elementów do formatowania za pomocą CSS (np. w celu zmiany koloru albo rozmiaru tekstu) i przetwarzania za pomocą JavaScriptu (np. gdy element zostanie kliknięty, strona ma zostać przekręcona do góry nogami).

Na razie w elemencie `div` jeszcze niczego nie ma, ale zanim coś do niego dodamy, sprawdzimy, czy wszystko działa tak, jak powinno. Zapisz plik *index.html* i uruchom przeglądarkę internetową. Aby otworzyć w niej swój plik, wpisz w pasku adresu jego adres, przeciągnij plik na pasek adresu przeglądarki albo kliknij go dwukrotnie myszą.

Gdy plik zostanie otwarty, w oknie przeglądarki powinien pojawić się widok pokazany na rysunku 1.1. Zwróć uwagę, że tytuł karty został ustawiony na *Quiz*, ponieważ to słowo wpisaliśmy w elemencie `title`.



Rysunek 1.1. Plik HTML otwarty w przeglądarce Chrome

Jeśli nie masz przeglądarki Chrome ani Firefoksa, to powinieneś je sobie teraz obie zainstalować. Ukazują różne problemy, jakie mogą występować podczas tworzenia gier przy użyciu HTML5, i obu będziesz często używać. Przeglądarki te nie są w tej książce traktowane jako idealne zamienniki.

Teraz dodamy trochę pytań w elemencie `div`. Widać je na listingu 1.2, gdzie zostały wyróżnione pogrubieniem. Jest to dość długi fragment kodu, ale znajduje się w nim wiele powtarzających się części. Jeśli nie masz chęci tego wszystkiego przepisywać, możesz to skopiować z pliku *quiz/po_recepturze1/index.html*.

Listing 1.2. Pytania quizu

```
<div id="quiz">
  <div id="question1">
    <div class="question">Który z tych typów plików nie jest używany do tworzenia
    ↪stron internetowych?</div>
    <input type="radio" name="question1" value="a"/>
    <label>.html</label>
    <input type="radio" name="question1" value="b"/>
    <label>.exe</label>
    <input type="radio" name="question1" value="c"/>
    <label>.js</label>
    <input type="radio" name="question1" value="d"/>
    <label>.css</label>
  </div>
  <br />
  <div id="question2">
    <div class="question">Która para znaków jest używana do oznaczania obiektów
    ↪JavaScript?</div>
    <input type="radio" name="question2" value="a"/>
    <label>[]</label>
    <input type="radio" name="question2" value="b"/>
    <label>;;</label>
    <input type="radio" name="question2" value="c"/>
    <label>{}</label>
    <input type="radio" name="question2" value="d"/>
    <label>()</label>
  </div>
  <br />
  <div id="question3">
    <div class="question">Krety są...</div>
    <input type="radio" name="question3" value="a"/>
    <label>wszystkożerne</label>
    <input type="radio" name="question3" value="b"/>
    <label>urocze</label>
    <input type="radio" name="question3" value="c"/>
```

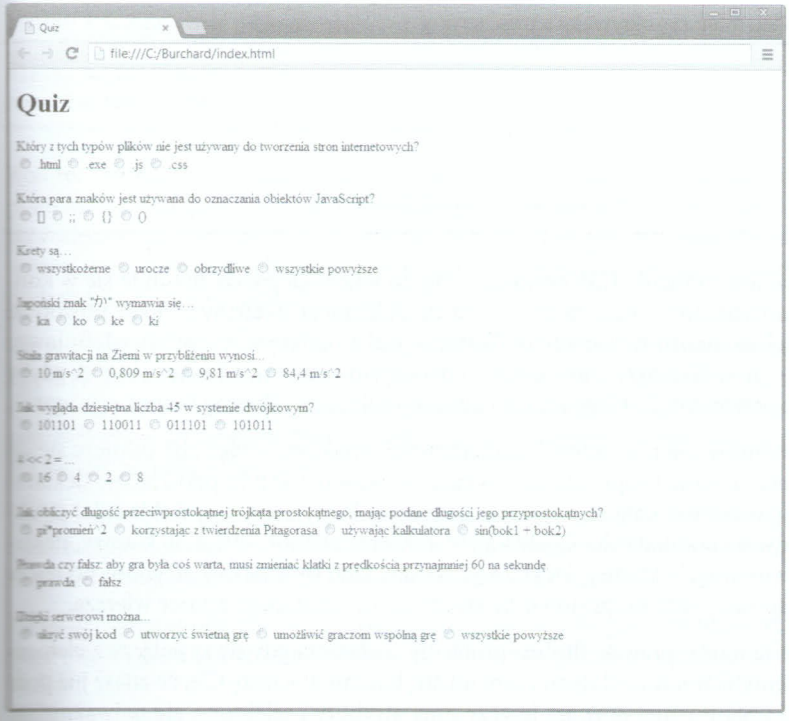
```
  <label>obrzydliwe</label>
  <input type="radio" name="question3" value="d"/>
  <label>wszystkie powyższe</label>
</div>
<br />
<div id="question4">
  <div class="question">Japoński znak "か" wymawia się...</div>
  <input type="radio" name="question4" value="a"/>
  <label>ka</label>
  <input type="radio" name="question4" value="b"/>
  <label>ko</label>
  <input type="radio" name="question4" value="c"/>
  <label>ke</label>
  <input type="radio" name="question4" value="d"/>
  <label>ki</label>
</div>
<br />
<div id="question5">
  <div class="question">Stała grawitacji na Ziemi w przybliżeniu
  ↪wynosi...</div>
  <input type="radio" name="question5" value="a"/>
  <label>10 m/s^2</label>
  <input type="radio" name="question5" value="b"/>
  <label>0,809 m/s^2</label>
  <input type="radio" name="question5" value="c"/>
  <label>9,81 m/s^2</label>
  <input type="radio" name="question5" value="d"/>
  <label>84,4 m/s^2</label>
</div>
<br />
<div id="question6">
  <div class="question">Jak wygląda dziesiętna liczba 45 w systemie
  ↪dwójkowym?</div>
  <input type="radio" name="question6" value="a"/>
  <label>101101</label>
  <input type="radio" name="question6" value="b"/>
  <label>110011</label>
  <input type="radio" name="question6" value="c"/>
  <label>011101</label>
  <input type="radio" name="question6" value="d"/>
  <label>101011</label>
</div>
<br />
<div id="question7">
  <div class="question">4 << 2 = ...</div>
  <input type="radio" name="question7" value="a"/>
  <label>16</label>
  <input type="radio" name="question7" value="b"/>
  <label>4</label>
  <input type="radio" name="question7" value="c"/>
  <label>2</label>
  <input type="radio" name="question7" value="d"/>
  <label>8</label>
</div>
<br />
<div id="question8">
  <div class="question">Jak obliczyć długość przeciwprostokątnej trójkąta
  ↪prostokątnego, mając podane długości jego przyprostokątnych? </div>
```

```
        <input type="radio" name="question8" value="a"/>
        <label>pi*promień^2</label>
        <input type="radio" name="question8" value="b"/>
        <label>korzystając z twierdzenia Pitagorasa</label>
        <input type="radio" name="question8" value="c"/>
        <label>używając kalkulatora</label>
        <input type="radio" name="question8" value="d"/>
        <label>sin(bok1 + bok2)</label>
      </div>
      <br />
      <div id="question9">
        <div class="question">Prawda czy fałsz: aby gra była coś warta, musi zmieniać
        ↪klatki z prędkością przynajmniej 60 na sekundę.</div>
        <input type="radio" name="question9" value="a"/>
        <label>prawda</label>
        <input type="radio" name="question9" value="b"/>
        <label>fałsz</label>
      </div>
      <br />
      <div id="question10">
        <div class="question">Dzięki serwerowi można...</div>
        <input type="radio" name="question10" value="a"/>
        <label>ukryć swój kod</label>
        <input type="radio" name="question10" value="b"/>
        <label>utworzyć świetną grę</label>
        <input type="radio" name="question10" value="c"/>
        <label>umożliwić graczom wspólną grę</label>
        <input type="radio" name="question10" value="d"/>
        <label>wszystkie powyższe</label>
      </div>
    </div>
...
```

Wszystkie pytania w tym quizie mają taką samą ogólną strukturę. Różnią się natomiast numerami, treścią oraz możliwościami do wyboru. Przyjmijmy, że interesuje nas tylko pierwsze pytanie. Znajduje się ono w elemencie `div` o identyfikatorze (`id`) `question1`. Identyfikator jest niepowtarzalny i można go będzie później użyć do różnych celów. Ten element `div` zawiera samo pytanie i cały blok odpowiedzi. W nim jest zagnieżdżony kolejny element `div` zawierający tylko samo pytanie. Ma on przypisaną klasę (`class`) `question`. Przypomnę, że za pomocą klasy, podobnie jak nazwy elementu i identyfikatora, można się później odwoływać do elementu. Najważniejszą różnicą między klasą a identyfikatorem jest to, że identyfikator nie może powtarzać się na stronie, a liczba klas jest nieograniczona.

Dalej znajduje się element `input` z trzema atrybutami. Atrybut `type="radio"` oznacza, że został utworzony przycisk radiowy. Jeśli nie wiesz, jak on wygląda, spójrz na rysunek 1.2. Drugi atrybut to `name`. Każda odpowiedź w zestawie musi mieć inną wartość tego atrybutu. Atrybut `value` określa, co jest przesyłane jako wartość elementu po zatwierdzeniu formularza HTML. Podobnie przesyłana jest zawartość pola tekstowego. My nie będziemy zatwierdzać formularzy, ale będziemy korzystać z tych wartości do sprawdzania odpowiedzi przy użyciu JavaScriptu. Poznałeś już elementy wymagające i niewymagające znacznika zamykającego. Natomiast omawiany element `input` ma zakończenie `/>` oznaczające, że sam się zamyka.

Elementy `label` służą do oznaczania tekstu znajdującego się poza elementami `input`. Ich głównym zadaniem jest przeniesienie fokusu na odpowiednie pola wejściowe w reakcji na ich klik-



Rysunek 1.2. Pytania i odpowiedzi quizu

nięcie. Nie ma tego w przykładowym kodzie, ale jeśli chcesz, możesz nadać każdej odpowiedzi niepowtarzalny identyfikator, np. `id="question-10-answer-b"`, oraz użyć go w atrybucie `for` odpowiedniej etykiety, np. `<label for="question-10-answer-b">`.

Między każdą parą pytań znajduje się znacznik `<br />`, w którym ukośnik oznacza, że jest to samozamykający się element. Element `br` służy do rozsuwania elementów w pionie. Wysokość tej pustej przestrzeni jest zależna od przeglądarki i dlatego jeśli układ elementów jest ważny (w większości przypadków jest, ale tutaj akurat nie), należy zamiast tego elementu używać własności CSS.

Jeśli wszystko poszło zgodnie z planem, to po otwarciu pliku w przeglądarce powinieneś zobaczyć widok pokazany na rysunku 1.2.

Receptura. Ukrywanie i pokazywanie quizu

W grach często występują różnego rodzaju blokady, np. zablokowane postaci, niedostępne plansze albo ukryte poziomy. Tutaj mamy zablokowane pytania. Może się wydawać, że to przestarzałe techniki, ale to tylko złudzenie. Nie chciałbyś przechodzić wszystkich plansz gry Mario naraz, prawda? To samo dotyczy quizu. Gdyby zawierał 100 pytań, to lepiej byłoby nie wyświetlać ich wszystkich jednocześnie.

Jak można zablokować treść? Jest wiele możliwości, wśród których można wymienić np. umieszczenie jej grupami na różnych stronach. Jednak dla uproszczenia w tym przypadku użyjemy kodu CSS, aby ukryć część treści strony. W związku z tym w folderze zawierającym plik *index.html* musimy utworzyć plik o nazwie *main.css* i zawartości pokazanej na listingu 1.3.

Listing 1.3. Zawartość pliku main.css ukrywająca treść strony

```
#quiz{
  display:none;
}
body{
  margin-left:50px;
}
```

Tekst `#quiz` oznacza, że jest to reguła CSS odnosząca się do wszystkiego, co znajduje się w kontenerze, np. `div`, o identyfikatorze (`id`) `quiz`. Użyta tu deklaracja `display: none` powoduje ukrycie całej zawartości elementu `div` o identyfikatorze `quiz`. Gdybyśmy chcieli zdefiniować właściwości elementu o identyfikatorze `inny-quiz`, to użylibyśmy selektora `#inny-quiz`. A gdyby interesował nas element przypisany do klasy `quiz`, to zamiast znaku # użylibyśmy kropki, np. `.quiz`.

Przed selektorami elementów nie ma żadnych dodatkowych znaków, a więc aby odnieść się do elementu `body`, nie trzeba używać kropki ani krzyżyka. Zastosowana w tym przykładzie deklaracja `margin-left:50px;` przesuwa całą stronę nieco w prawo. Przyjrzyjmy się dokładniej strukturze tych dwóch bloków formatujących. Każdy z nich zawiera selektor, otwarcie klamry, deklaracje stylistyczne oraz zamknięcie klamry. Deklaracja stylu składa się z nazwy atrybutu po lewej, dwukropka, wartości tego atrybutu po prawej oraz średnika oznaczającego koniec wiersza.

Początkującym składnia ta może sprawiać drobne problemy, zwłaszcza gdy się ją połączy z elementami HTML oraz ich identyfikatorami, klasami i innymi atrybutami. Pocieszę Cię, że znasz już podstawy języków HTML i CSS. Później poznasz jeszcze inne atrybuty i selektory, ale najważniejsze podstawy już znasz. Pewnie nie raz popełnisz jakiś błąd, np. użyjesz krzyżyka zamiast kropki albo odwrotnie, zapomnisz o końcowym średniku lub zamknięciu klamry itp. Nie przejmuj się jednak. Są to powszechnie występujące błędy, których nie potrafią się ustrzec nawet zawodowcy. Jeśli coś nie będzie działać, zastanów się przez chwilę i dokładnie przeczytaj napisany przez siebie kod.

Jeśli teraz zapiszesz pliki i otworzysz plik *index.html* w przeglądarce, to zobaczysz, że strona wygląda tak jak na rysunku 1.1, chociaż jest odrobinę wcięta w prawo.

Receptura. Przywracanie pytań do widoku

Wszystkie pytania zniknęły i trzeba je jakoś przywrócić do widoku. Możesz to zrobić, dodając pakiety użyte w każdym z kolejnych rozdziałów. Każdy pakiet spowoduje pojawienie się jednego pytania.

Zanim zaczniemy ładować pakiety, musimy sprawdzić, czy w ogóle możemy ładować kod JavaScript. Prawie na samym dole pliku *index.html* wpisz kod wyróżniony pogrubieniem na listingu 1.4.

Listing 1.4. Ładowanie pierwszego zewnętrznego pliku JavaScript

```
...
    <script src="game.js"></script>
  </body>
</html>
```

W ten sposób ładujemy na stronę plik JavaScript o nazwie *game.js*. Oczywiście musimy też go utworzyć. Utwórz plik o nazwie *game.js* w tym samym folderze, w którym znajdują się pliki *main.css* i *index.html*, oraz wpisz w nim kod widoczny na listingu 1.5.

Listing 1.5. Zawartość pliku game.js

```
alert('Witaj, świecie');
console.log('Witaj, świecie');
```

Kod ten drukuje informacje w dwóch miejscach. Pierwsze z nich stanie się oczywiste po otwarciu pliku *index.html*, ponieważ jest to wyskakujące okienko alertu. Natomiast instrukcja `console.log` powoduje wysłanie tekstu do konsoli JavaScript, która jest niezbędnym narzędziem dla każdego webmastera. Jeśli potrzebujesz pomocy na temat podstaw obsługi konsoli JavaScript, zajrzyj do dodatku B „Kontrola jakości".

Teraz dodamy bibliotekę jQuery. Najprościej jest w tym celu wejść na stronę *http://jquery.com* i pobrać bibliotekę na swój dysk w dowolny sposób. Ja po prostu kliknąłem największy i najbardziej efektowny przycisk znajdujący się na stronie, aby przejść na stronę zawierającą kod tej biblioteki. Następnie go skopiowałem i wkleiłem do utworzonego własnoręcznie pliku o nazwie *jquery.js*. Na koniec zapisałem ten plik.

Na stronie jQuery można też pobrać plik biblioteki w tradycyjny sposób. Nieważne, jak ją zdobędziesz, pamiętaj tylko, aby umieścić ją w odpowiednim folderze na swoim dysku (w tym samym, w którym znajdują się pliki *index.html*, *main.css* i *game.js*).

Po umieszczeniu pliku w odpowiednim miejscu na dole pliku *index.html* dodaj kod wyróżniony pogrubieniem na listingu 1.6. Upewnij się, że nazwa pliku jest taka sama jak nazwa wpisana w tym kodzie.

Listing 1.6. Dodanie biblioteki jQuery do pliku index.html

```
    <script src="jquery.js"></script>
    <script src="game.js"></script>
  </body>
</html>
```

Jeśli swojemu plikowi nadałeś inną nazwę niż *jquery.js*, pamiętaj, żeby zmienić ją także w powyższym kodzie.

Teraz przydałoby się nieco dostosować arkusz stylów. Wcześniej zadziałaliśmy trochę zbyt agresywnie. Teraz to zmienimy i zamiast ukrywać wszystkie pytania naraz, schowamy każde z nich osobno przy użyciu kodu widocznego na listingu 1.7.

Listing 1.7. Ukrywanie pytań, nie całego quizu

```
body{
  margin-left:50px;
}

#question1, #question2, #question3, #question4, #question5,
#question6, #question7, #question8, #question9, #question10{
  display:none;
}
```

Został usunięty selektor `#quiz`, a w jego miejsce wstawiliśmy listę rozdzielanych przecinkami selektorów identyfikatorów pytań. Można też było przypisać wszystkim pytaniom wspólną klasę i ukryć je wszystkie przy użyciu selektora kropki. Ale warto wiedzieć, że można też tworzyć takie listy selektorów jak powyższa.

Po ukryciu pytań za pomocą CSS możemy je odblokować przy użyciu jQuery. W tym celu musimy zmienić kod znajdujący się w pliku *game.js* na pokazany na listingu 1.8. Należy nim zastąpić poprzednią zawartość tego pliku.

Listing 1.8. Kod powodujący wyświetlenie pierwszego pytania, jeżeli jest załadowana biblioteka jQuery

```
if(jQuery){
  $("#question1").show();
};
```

Znajdująca się w pierwszym wierszu instrukcja warunkowa sprawdza, czy jest załadowana biblioteka jQuery. Jeśli tak, następuje wykonanie drugiego wiersza kodu. W tym wierszu jest użyta funkcja jQuery `$`, której przekazujemy selektor CSS `#question1` w cudzysłowie i nawiasie. Następnie wykonujemy funkcję `show` w celu zamiany deklaracji `display:none` pierwszego pytania na `display:block`.

Jeśli teraz zapiszesz pliki i otworzysz stronę *index.html* w przeglądarce, zobaczysz, że pojawiło się pierwsze pytanie.

Receptura. Lista zakupów

W tej recepturze zaimportujemy na naszą stronę jeszcze dziewięć dodatkowych plików. Pewnie zastanawiasz się, dlaczego warunkiem wyświetlenia pytań ma być załadowanie jakichś plików. Wielu osobom może się wydawać, że pobieranie plików na dysk, a następnie dołączanie ich do innych plików jest bez sensu. Jednak umiejętność korzystania z kodu napisanego przez inne osoby jest bardzo ważna. Niewiele projektów tworzy się zupełnie od podstaw, a nauczenie się tworzenia gier poprzez „stawanie na ramionach olbrzymów" jest naprawdę warte zachodu. Ponadto w tej części rozdziału zrobisz przegląd, jakiego rodzaju plików będziesz używać w dalszych częściach kursu.

Jeżeli wiesz, jak się dołącza pliki JavaScript do systemu, i dobrze znasz metody kontroli wersji, to pozostałe podrozdziały będą dla Ciebie jedynie powtórką. Możesz je tylko przejrzeć, a nawet pominąć, jeśli chcesz.

Mając załatwioną sprawę z najważniejszą w tym rozdziale biblioteką, jQuery, możemy udać się na dalsze zakupy. Jeśli masz ochotę na małą przygodę, to możesz wszystkie biblioteki pobrać z ich stron, których adresy znajdziesz w dodatku C „Zasoby". Ale możesz też je wszystkie znaleźć w folderze *po_recepturze4* w katalogu plików do tego rozdziału. Nie zapomnij tylko umieścić wszystkich plików w tym samym folderze, w którym znajduje się plik *index.html*.

Po zdobyciu wszystkich plików w ten czy inny sposób Twój system plików powinien wyglądać tak jak na rysunku 1.3.

Teraz możesz rozpocząć dołączanie plików JavaScript do strony, dodając wiersze pogrubione na listingu 1.9 do pliku *index.html*.

ARTYKUŁY SPOŻYWCZE

1. ***jquery.js***: ten plik już masz. Jest używany w kilku innych rozdziałach do wybierania elementów na stronie i manipulowania nimi.
2. ***impress.js***: w rozdziale 3. „Impreza" tego narzędzia do tworzenia prezentacji (podobnego do PowerPointa, ale w JavaScripcie) użyjemy do zarządzania „stronami" interaktywnej gry.
3. ***atom.js***: jest to jeden z najmniejszych dostępnych silników gier (zawiera tylko 203 nieskompresowane wiersze kodu CoffeeScript). Skryptu tego użyjemy do budowy gry imprezowej.
4. ***easel.js***: skrypt udostępniający udoskonalony interfejs do API kanwy, którego będziemy używać przy rysowaniu puzzli.
5. ***melon.js***: tego silnika użyjemy do budowy platformówki w rozdziale 5.
6. ***yabble.js***: w grze symulującej walkę wykorzystamy tę bibliotekę do załadowania silnika *game.js* (nie mylić z plikiem *game.js* użytym w tym rozdziale i innych).
7. ***jquery.gamequery.js***: wtyczka do jQuery będąca silnikiem gier. Użyjemy jej do utworzenia strzelanki, w której poruszamy się z boku ekranu.
8. ***jaws.js***: tego wszechstronnego silnika gier (i staroświeckiej trygonometrii) użyjemy do budowy typowej gry FPS.
9. ***enchant.js***: japoński silnik gier o bogatej funkcjonalności i doskonałej obsłudze urządzeń mobilnych. Użyjemy go do budowy gry RPG w rozdziale 9. „RPG".
10. ***crafty.js***: rozbudowany i bardzo dobrze obsługiwany silnik gier, którego użyjemy do budowy gry RTS (gdybym miał wybrać jeden silnik, który zabrałbym na bezludną wyspę, to możliwe, że wybrałbym właśnie ten).

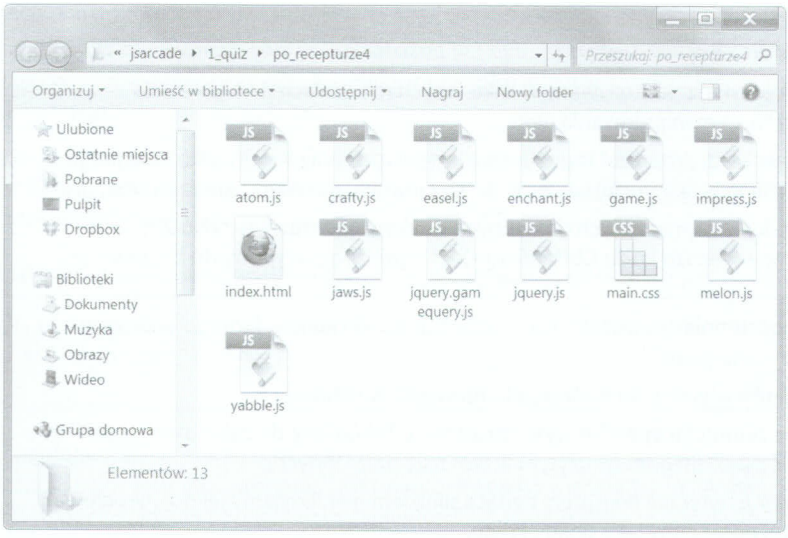
UWAGA

Jeśli przeczytałeś dodatek C, to zapewne zauważyłeś, że wszystkie wymienione pliki można też znaleźć w serwisie GitHub. Pliki z tego serwisu można pobierać na trzy sposoby. Po pierwsze, można pobrać cały projekt w formacie ZIP, wypakować pliki i użyć tych, które są potrzebne.

Po drugie, można przejrzeć zawartość projektu, kliknąć wybrany plik, skopiować jego zawartość, a następnie wkleić ją do nowo utworzonego pliku na własnym dysku. Może się wydawać, że to dużo zachodu, ale w istocie ta metoda jest naprawdę szybka.

Trzecia możliwość jest nieco bardziej skomplikowana, ale może zaowocować ułatwieniem pracy w przyszłości. Polega ona na zainstalowaniu programu Git w komputerze, pobraniu (sklonowaniu) projektu oraz przejściu do folderu tego rozdziału w celu pobrania plików. Możesz pracować bezpośrednio w tym folderze albo skopiować z niego potrzebne pliki.

Git to system kontroli wersji umożliwiający śledzenie zmian w plikach. Natomiast GitHub to portal internetowy, w którym osoby używające programu Git (wielu programistów z różnych krajów) mogą przechowywać własne projekty i znajdować projekty innych osób. Publicznie udostępnione projekty można przechowywać za darmo. Gorąco polecam skorzystanie z tej możliwości. Najlepszy poradnik instalacji programu Git znajduje się pod adresem *help.github.com/articles/set-up-git*.



Rysunek 1.3. Katalog zawierający wszystkie potrzebne pliki JavaScript

Listing 1.9. Dodawanie plików JavaScript do strony index.html

```
    <script src="jquery.js"></script>
    <script src="impress.js"></script>
    <!-- To jest potrzebne do działania skryptu atom.js -->
    <canvas></canvas>
    <script src="atom.js"></script>
    <script src="easel.js"></script>
    <script src="melon.js"></script>
    <script src="yabble.js"></script>
    <script src="jquery.gamequery.js"></script>
    <script src="jaws.js"></script>
    <script src="enchant.js"></script>
    <script src="crafty.js"></script>
    <script src="game.js"></script>
  </body>
</html>
```

Upewnij się, że nazwy plików dołączanych do strony zgadzają się z nazwami wpisanymi w elementach `script`. Pliki JavaScript do stron HTML zazwyczaj dołącza się właśnie przy użyciu elementu `script`. W całym tym kodzie znajduje się tylko jedna nietypowa rzecz — element `canvas` umieszczony między skryptem *atom.js* a komentarzem `<!-- -->`. Element ten jest potrzebny do działania biblioteki *atom.js*. Większość silników gier uruchamia się poprzez wywołanie funkcji inicjującej albo wskazanie konkretnego elementu `canvas`, który ma zostać wykorzystany. Jednak skrypt *atom.js* automatycznie szuka elementu `canvas` natychmiast, gdy tylko zostanie dołączony do strony. Zamiast z nim walczyć (tzn. edytować plik *atom.js*), lepiej jest dać mu to, czego chce. Znaki `<!-- -->` oznaczają komentarz HTML. Komentarze służą do wpisywania na stronie notatek przeznaczonych dla nas samych lub innych osób, które są ignorowane przez przeglądarki. Należy jednak pamiętać, że może je przeczytać każdy użytkownik, który zajrzy do kodu źródłowego strony. Jeśli nie wiesz, o co mi chodzi, przeczytaj dodatek B.

Teraz w pliku *game.js* przywrócimy do widoku pozostałe pytania naszego quizu. W tym celu należy dodać wiersze oznaczone na listingu 1.10 pogrubieniem.

Listing 1.10. Przywrócenie pozostałych pytań do widoku

```
if(jQuery){
  $("#question1").show();
};
if(impress){
  $("#question2").show();
};
if(atom){
  $("#question3").show();
};
if(createjs){
  $("#question4").show();
};
if(me){
  $("#question5").show();
};
if(require){
  $("#question6").show();
};
if($().playground){
  $("#question7").show();
};
if(jaws){
  $("#question8").show();
};
if(enchant){
  $("#question9").show();
};
if(Crafty){
  $("#question10").show();
};
```

Efekt dodania każdej z tych instrukcji jest od razu widoczny w postaci pojawienia się nowego obiektu na stronie. Jedynym wyjątkiem w tym bloku kodu jest test `playground` dotyczący pytania 7. gameQuery to rozszerzenie jQuery, a więc jego funkcje bazują na funkcjach tej biblioteki. Nie ma własnego rdzennego obiektu i dlatego trzeba sprawdzić dostępność funkcji `playground` w obiekcie `$()` jQuery.

OSTRZEŻENIE

TO NIE JEST TWÓJ KOD. Programiści piszący kod zazwyczaj chcą mieć odrobinę kontroli nad sposobem jego używania przez innych. Kontrolę tę sprawują poprzez dołączenie do kodu licencji. Nie oznacza to, że takich programów nie można albo nie należy używać. Niektóre licencje zabraniają tylko używać kodu w celach komercyjnych, inne wymagają podania gdzieś nazwiska autora skryptu, a jeszcze inne są tylko po to, aby skrypt mógł być zawsze używany. Szczegółowy opis kwestii licencjonowania oprogramowania wykracza poza zakres tej książki, ale jeśli przeczytasz licencje użytych w niej bibliotek lub poczytasz o licencjach Creative Commons, GPL, BSD i MIT, to będziesz się orientować, jak inni zapatrują się na kwestię otwartości oprogramowania. To samo dotyczy obrazów, plików dźwiękowych i innych typów treści.

Jeśli teraz zapiszesz plik *index.html* i otworzysz go w przeglądarce internetowej, to zobaczysz cały quiz, chociaż nie będzie on reagował na kliknięcia. Powodem tego jest rozciągnięcie na powierzchni całej strony elementu canvas, który jak niewidoczna płachta przykrywa wszystko, blokując dostęp do elementów strony. Problem ten rozwiążemy, dodając prostą regułę CSS (pogrubiony kod na listingu 1.11).

Listing 1.11. Kod CSS ukrywający element canvas

```
body{
  margin-left:50px;
}
#question1, #question2, #question3, #question4, #question5,
#question6, #question7, #question8, #question9, #question10{
display:none;
}
canvas{
  display:none;
}
```

Receptura. Które odpowiedzi są poprawne

Poprawne odpowiedzi można by było oznaczyć, dodając do nich klasę `correct`, ale to zbyt proste rozwiązanie zarówno pod względem implementacji, jak i ryzyka podejrzenia odpowiedzi przez użytkownika. Wszystko, co znajduje się w tych plikach, nawet komentarze, jest widoczne dla użytkownika, który jeśli nie będzie znał odpowiedzi, będzie mógł ją podejrzeć. Aby trochę utrudnić oszukiwanie osobom znającym się na programowaniu i uniemożliwić tym, które się nie znają, do sprawdzania odpowiedzi można użyć słabej funkcji mieszającej.

Funkcja mieszająca to funkcja pobierająca wartość i przekształcająca ją w inną wartość. Jej zaletą w tym przypadku jest łatwość, z jaką można odkryć pierwotną wartość, mając wynik mieszania.

Zanim ją napiszemy, najpierw utworzymy styl informujący w widoczny sposób, że wszystkie odpowiedzi są poprawne. Styl ten, zapisany w pliku *main.css*, jest pokazany na listingu 1.12 i wyróżniony pogrubieniem.

Listing 1.12. Styl włączany, gdy użytkownik poprawnie odpowie na wszystkie pytania

```
body{
  margin-left:50px;
}
#question1, #question2, #question3, #question4, #question5,
#question6, #question7, #question8, #question9, #question10{
display:none;
}
canvas{
  display:none;
}
.correct{
  background-color:#24399f;
  color:white;
}
```

Dodana reguła definiuje niebieskie tło i biały tekst dla elementów należących do klasy `correct`. Klasę tę można dodać do quizu, gdy użytkownik poprawnie odpowie na wszystkie pytania. W przedszkolu albo gdzieś indziej może słyszałeś o kolorze białym, ale kolor o nazwie `#24399f` raczej rzadko pojawia się w codziennych konwersacjach, nawet wśród absolwentów większości kierunków technicznych. Jest to definicja koloru w formacie RGB (ang. *red, green, blue* — czerwony, zielony, niebieski). Dwie pierwsze cyfry określają wartość czerwieni, następne dwie — wartość zieleni, a ostatnie dwie — ilość niebieskiego.

Ale chwileczkę, ostatnia cyfra to litera `f`. Litera to przecież nie cyfra. W istocie w dziesiętnym systemie liczbowym nie ma takiej cyfry. Ale gdybyśmy używali systemu dziesiętnego, to mielibyśmy do dyspozycji tylko 100 (0 – 9 i 0 – 9, czyli 10·10) wartości dla każdej z barw składowych. Ktoś uznał, że to za mało jak na sieć, i dlatego używamy systemu szesnastkowego, w którym dla każdej barwy RGB jest dostępnych 256 (16·16) odcieni. Istnieje też ograniczony zbiór nazw kolorów i można np. napisać `white` albo `#ffffff` oraz `black` albo `#000000`. Przy okazji ktoś inny kiedyś pomyślał sobie, że tych cyfr czasami jest za dużo, i dlatego powtarzające się cyfry można zredukować do trzech, np. kolor czarny można zapisać jako `#000`, a biały — `#fff`.

Po dodaniu kodu CSS pozostaje jeszcze zmienić coś w pliku *index.html*. Znacznik otwierający `<body>` zamień na znacznik oznaczony pogrubieniem na listingu 1.13.

Listing 1.13. Dodanie procedury obsługi kliknięcia do elementu body w pliku index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Quiz</title>
    <link rel="stylesheet" type="text/css" href="main.css">
  </head>
  <body onclick="checkAnswers();">
```

Zamiast zwykłego znacznika `<body>` mamy teraz znacznik z atrybutem `onclick` zawierającym łańcuch kodu JavaScript w cudzysłowie. Jeśli dziwi Cię słowo „łańcuch", przeczytaj dodatek A i wróć potem do tego miejsca. Łańcuch znajdujący się w tym atrybucie `onclick` powoduje wywołanie funkcji `checkAnswers` za każdym razem, gdy zostanie kliknięty jakiś element na stronie. Zwróć uwagę na nawias, który oznacza, że jest to wywołanie funkcji. Gdyby go nie było, po prostu odnosilibyśmy się do funkcji, ale byśmy jej nie wywoływali.

Na listingu 1.14 znajduje się ostatni przykład kodu prezentowany w tym rozdziale. Jest to treść opisanej powyżej funkcji. Pogrubiony kod z tego listingu można umieścić na początku pliku *game.js*, zaraz za testem obecności biblioteki jQuery a instrukcją powodującą wyświetlenie pierwszego pytania.

Listing 1.14. Sprawdzenie odpowiedzi

```
$(document).ready(function(){
  var checkAnswers = function(){
    var answerString = "";
    var answers = $(":checked");
    answers.each(function(i) {
      answerString = answerString + answers[i].value;
    });
    $(":checked").each(function(i) {
```

```
      var answerString = answerString + answers[i].value;
    });
    checkIfCorrect(answerString);
  };
  var checkIfCorrect = function(theString){
    if(parseInt(theString, 16) === 811124566973){
      $("body").addClass("correct");
      $("h1").text("Wygrałeś!");
      $("canvas").show();
    }
  };
  $("#question1").show();
};
...
```

W pogrubionym kodzie znajdują się definicje dwóch funkcji. Pierwsza ma nazwę `checkAnswers` i tworzy pusty łańcuch, do którego będziemy dodawać kolejne odpowiedzi, gdy użytkownik będzie klikał przyciski radiowe. Po zakończeniu tej pętli zostaje wywołana druga funkcja, `checkIfCorrect`, porównująca otrzymany łańcuch z długą liczbą. Skąd wzięła się ta liczba?

Przypomnij sobie szesnastkowe wartości kolorów CSS. Można w nich używać cyfr od 0 do f. To oznacza, że litery a – d, będące odpowiedziami w naszym quizie, mogą być traktowane jak cyfry systemu szesnastkowego (można je traktować jak liczby 10 – 13). Połączyłem je w jeden łańcuch, który następnie zamieniłem na format dziesiętny.

Jeśli wynik porównania jest pozytywny, dodajemy do elementu `body` klasę `correct`, co powoduje zmianę koloru tła i tekstu. Dodatkowo następuje zamiana tekstu elementu `h1` z `Quiz` na `Wygrałeś!`. Na zakończenie wykorzystujemy ukryty wcześniej element `canvas` do zablokowania możliwości używania myszy na ekranie. Normalnie w celu zablokowania możliwości używania elementów formularza na stronie posłużylibyśmy się funkcją jQuery `disable`, ale dzięki tej sztuczce znaleźliśmy zastosowanie dla elementu `canvas`, który w innym przypadku byłby całkiem bezużyteczny. Ponadto element ten można by było wykorzystać jako bazę gry opartej na silniku *atom.js* i wówczas udzielenie poprawnych odpowiedzi w quizie byłoby warunkiem rozpoczęcia gry.

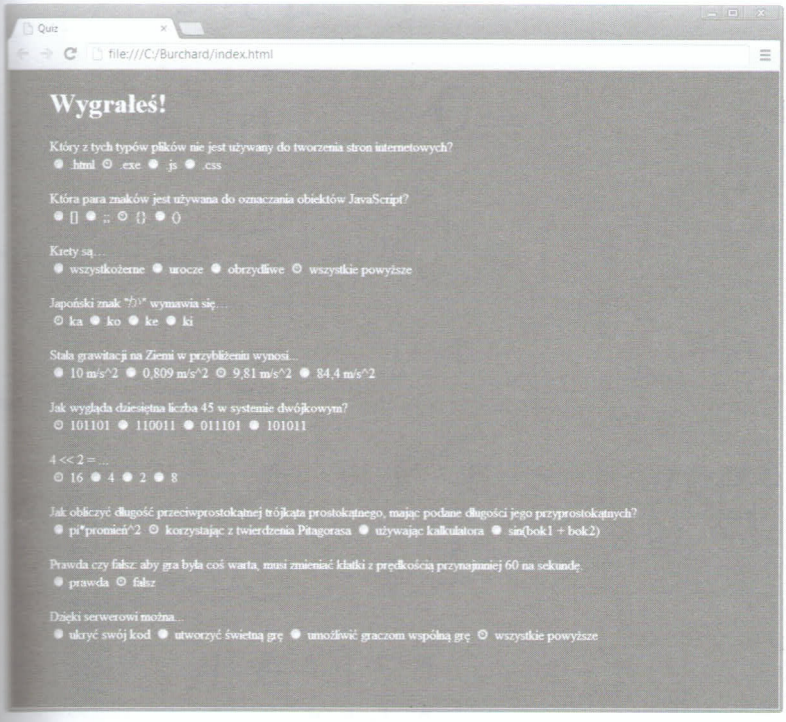
Po wykonaniu wszystkich opisanych czynności oraz zapisaniu wszystkich plików i otwarciu w przeglądarce pliku *index.html* powinieneś zobaczyć stronę pokazaną na rysunku 1.4.

Podsumowanie

W tym rozdziale utworzyliśmy prosty quiz z pytaniami mającymi związek z każdym rozdziałem tej książki. Wszystkie pytania były zablokowane i aby je odblokować, trzeba było dołączyć do strony pliki JavaScript, które będą używane w różnych rozdziałach. Do sprawdzania odpowiedzi użyliśmy prostej funkcji mieszającej zamieniającej wartości szesnastkowe reprezentujące odpowiedzi na długą wartość dziesiętną.

Podczas tworzenia tej gry poznałeś podstawy technologii HTML, CSS, jQuery, Git oraz dowiedziałeś się o istnieniu niektórych licencji na oprogramowanie. Ponadto poznałeś nazwy silników gier i innych bibliotek, których będziemy używać we wszystkich pozostałych rozdziałach.

Jeśli chcesz trochę poćwiczyć, możesz spróbować utworzyć drugą stronę z pytaniami, która będzie odblokowywana dopiero wtedy, gdy użytkownik udzieli prawidłowych odpowiedzi na pierwszej. W rozdziale 2. znajduje się opis jednej z możliwości wyświetlania bardziej dynamicznych



Rysunek 1.4. Wszystkie poprawne odpowiedzi z informacją o wygranej

informacji, a więc możesz w nim poszukać inspiracji. Ponadto na zakończenie gry przykryliśmy wszystko elementem canvas. Możesz umieścić na nim inną grę. Skrypt *atom.js* kontroluje go i czeka, aż wrócisz po lekturze rozdziału 3., aby coś na nim dodać, gdy już będziesz wiedział, jak to zrobić.

Jeśli treść tego rozdziału była dla Ciebie trudna do zrozumienia, przestudiuj go jeszcze raz wraz z dodatkiem A. A jeśli nie znalazłeś w nim nic nowego, to nie przejmuj się. Od rozdziału 2. zaczynamy prawdziwą zabawę, a w rozdziale 7. idziemy już na całość.