7.2 <обработка событий мыши>

Мы уже говорили о том, что одним из самых привлекательных нововведений HTML 4.0 является возможность динамически изменять страницы и «реагировать» на пользовательские действия. Давайте рассмотрим, как такая «реакция» может осуществляться.

Помните, как в примерах из главы 4 мы изменяли цвет гиперссылки при наведении на неё мыши? Это происходило с помощью псевдокласса :hover. Однако этот псевдокласс пока что определён только для тэга <A>. А как быть, если мы хотим изменить цвет обычного текста при наведении на него мыши?

Рассмотрим, как это делается. Допустим, мы написали небольшую тестовую страничку:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
     <TITLE>Обработка событий мыши</TITLE>
</HEAD>
<BODY>
Этот текст не изменит свой цвет.
Этот текст изменит свой цвет, если навести на него мышь!
Этот текст не изменит свой цвет.
</BODY>
</HTML>
```

Теперь давайте сделаем так, чтобы вторая строка этого текста действительно изменяла свой цвет при наведении мыши. Для начала давайте выделим её в отдельный блок:

```
<DIV>Этот текст изменит свой цвет, если навести на него
мышь!</DIV>
```

Для того, чтобы при наведении мыши что-нибудь произошло, нужно добавить обработчик событий onMouseOver:

```
<DIV onMouseOver="">Этот текст изменит свой цвет, если навести на
него мышь!</DIV>
```

Итак, обработчик добавлен, однако пока что он ничего не делает. В кавычки нужно поместить то действие, которое он должен выполнить. А что он должен сделать? Изменить цвет этого блока <DIV>, например, на красный. Доступ к свойствам текущего элемента осуществляется с помощью ключегвого слова this:

```
<DIV onMouseOver="this.style.color='red'">Этот текст изменит свой
цвет, если навести на него мышь!</DIV>
```

Если теперь открыть эту страничку в браузере, то при наведении мыши на вторую строку текста её цвет действительно изменится на красный[1]. Однако, изменившись, он так и останется красным. Для того, чтобы при убирании мыши с этой строки цвет изменился обратно на чёрный, добавим обработчик событий, реагирующий на убирание указателя мыши. Он называется onMouseOut:

```
<DIV onMouseOver="this.style.color='red'"
onMouseOut="this.style.color='black'">Этот текст изменит свой
цвет, если навести на него мышь!</DIV>
```

Теперь при наведении мыши на эту строку её цвет изменится на красный, а при убирании мыши — обратно на чёрный.

Можно также использовать и доступ по названию элемента. Например, если установить в этом блоке атрибут ID="text1", то можно будет написать так:

```
<DIV ID="text1" onMouseOver="text1.style.color='red'"
onMouseOut="text1.style.color='black'">Этот текст изменит свой
цвет, если навести на него мышь!</DIV>
```

При этом лучше использовать полную форму записи доступа через коллекцию document.all, как объяснялось в главе 6:

```
<DIV ID="text1" onMouseOver="document.all.text1.style.color='red'"
onMouseOut="document.all.text1.style.color='black'">Этот текст
изменит свой цвет, если навести на него мышь!</DIV>
```

Обратите внимание, что внутри кавычек расположен текст на JavaScript. Для того, чтобы не загромождать текст HTML-документа, можно заранее определить соответствующие функции в секции <HEAD>:

```
<HEAD>
     <TITLE>Обработка событий мыши</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function change() {
     document.all.text1.style.color="red";
}
function change2() {
     document.all.text1.style.color="black";
}
//-->
</SCRIPT>
</HEAD>
```

а при определении обработчиков событий писать только имена этих функций:

```
<DIV ID="text1" onMouseOver="change()" onMouseOut="change2()">Этот
текст изменит свой цвет, если навести на него мышь!</DIV>
```

1 В браузере Netscape версии 4 (и ниже) это не сработает.

Результат будет тем же, что и в прошлый раз.

Таким же образом можно изменить не только тот блок текста, на который мы навели мышь, но и любые другие элементы, если только присвоить имя. Например, если мы хотим одновременно с изменением цвета второй строки на красный изменять цвет третьей строки, скажем, на зелёный, достаточно будет сначала присвоить третьей строке имя:

```
<DIV ID="text2">Этот текст изменит свой цвет, если мышь навести на
вторую строку!</DIV>
```

а потом соответствующим образом изменить функции:

```
function change() {
     document.all.text1.style.color="red";
     document.all.text2.style.color="green";

}
function change2() {
     document.all.text1.style.color="black";
     document.all.text2.style.color="black";
}
```

Теперь при наведении мыши на вторую строку её цвет будет изменяться на красный, а цвет третьей строки — на зелёный.

К сожалению, здесь начинают сильно сказываться различия между браузерами. Доступ через document.all будет работать в Internet Explorer, но не сработает в Netscape. Для того, чтобы этот пример мог работать в Netscape 6, нужно доступ через document.all заменить на доступ через document.getElementById(). А как быть, если мы хотим, чтобы этот пример работал и в Internet Explorer, и в Netscape 6?

Такие вопросы обычно решаются непросто. Но в данном случае мы можем осуществить проверку на браузер, и в зависимости от её результата присвоить переменной text1 либо значение document.all.text1, либо значение document.getElementById('text1'). А затем в функциях замены цвета просто подставлять эту переменную. Точно то же самое можно проделать и с переменной text2. Только нужно не забывать заранее определить эти переменные:

```
var text1, text2;
function brws() {
     if (navigator.appName!="Netscape") {
          text1=document.all.text1;
          text2=document.all.text2;
          }
     else {
          text1=document.getElementById('text1');
          text2=document.getElementById('text2');
          }
}
```

Теперь необходимо сделать так, чтобы наша функция brws() выполнялась сразу после загрузки страницы. Для этого установим в тэг <BODY> обработчик событий, реагирующий на загрузку элемента. Он называется onLoad:

<BODY onLoad="brws()">

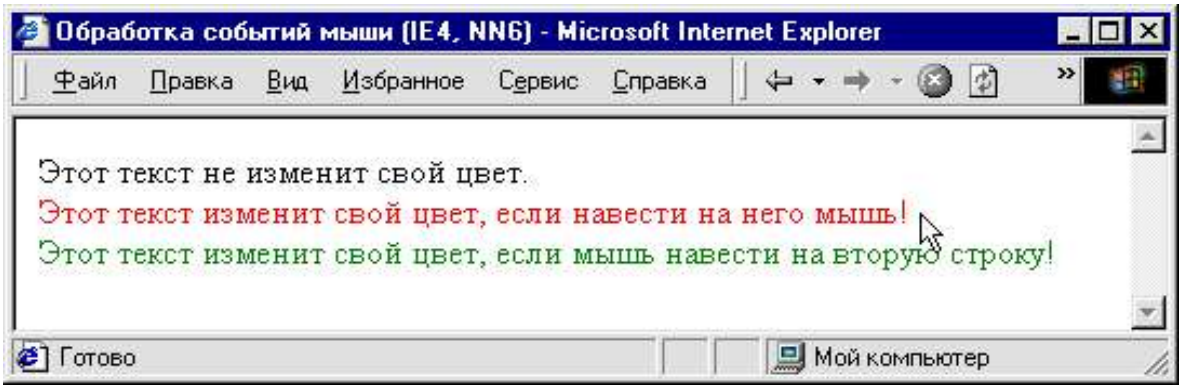
Посмотрим, что у нас получается в целом:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
     <TITLE>Обработка событий мыши (IE4, NN6)</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var text1, text2;
function brws() {
     if (navigator.appName!="Netscape") {
          text1=document.all.text1;
          text2=document.all.text2;
          }
     else {
          text1=document.getElementById('text1');
          text2=document.getElementById('text2');
          }
}
function change() {
     text1.style.color="red";
     text2.style.color="green";

}
function change2() {
     text1.style.color="black";
     text2.style.color="black";
}
//-->
</SCRIPT>
</HEAD>

<BODY onLoad="brws()">
Этот текст не изменит свой цвет.
<DIV ID="text1" onMouseOver="change()" onMouseOut="change2()">Этот
текст изменит свой цвет, если навести на него мышь!</DIV>
<DIV ID="text2">Этот текст изменит свой цвет, если мышь навести на
вторую строку!</DIV>
</BODY>
</HTML>
```



Результат вы можете видеть на рис.7.5. Теперь этот пример работает и в Internet Explorer, и в Netscape 6. Конечно, хотелось бы создавать такие страницы, которые бы работали по крайней мере в этих двух браузерах (потому что в Netscape версии 4 поддержка динамических страниц вообще очень слабая). Однако это не всегда возможно. В этой книге в дальнейшем все примеры будут ориентированы на Internet Explorer версии 4 и выше (если специально не оговорено обратное).

Теперь давайте рассмотрим такой пример. Предположим, вы разместили на веб-странице сказку, как на рис.3.11, с градиентным фоном. Однако вы беспокоитесь о том, что кому-то этот фон может мешать читать текст, и, заботясь о пользователе, хотите предусмотреть кнопку, при нажатии на которую фон бы становился равномерно зеленоватым, как на рис.3.10. Более того, для пользователей, предпочитающих читать чёрные буквы на белом фоне, вы хотите предусмотреть возможность смены цвета фона на белый.

Попробуем это реализовать. Возьмём в качестве исходного текст веб-страницы с рис.3.11, приведённый в главе 3. Для начала давайте в соответствии с требованиями HTML 4.0 заменим атрибуты BGCOLOR и BACKGROUND тэга <BODY> на соответствующие стилевые свойства:

```
<STYLE>
     BODY {    background-color: #BFFFBF;
               background-image: url('Images/grad1.jpg');
     }
</STYLE>
```

Что же касается тэга <BODY>, то ему желательно присвоить имя для облегчения доступа к его свойствам:

```
<BODY ID="doc">
```

Теперь давайте добавим сверху две кнопки — одну для выключения фонового рисунка, а другую — для смены цвета фона на белый:

```
<DIV ALIGN="center">
<INPUT TYPE="button" VALUE="Убрать фоновый рисунок">
<INPUT TYPE="button" VALUE="Сделать фон белым">
</DIV>
```

Кнопки созданы, но пока при нажатии на них ничего не произойдёт. Давайте теперь напишем функцию, убирающую фоновый рисунок. Для этого нужно всего лишь стилевому свойству background-image присвоить значение none:

```
function noBg() {
     document.all.doc.style.backgroundImage='none';
}
```

Обратите внимание, что в тексте на JavaScript (а не на языке CSS) нужно обязательно преобразовать название стилевого свойства с дефисом, как объяснялось выше.

Теперь нужно сделать так, чтобы наша функция noBg() выполнялась при щелчке мыши на первой из кнопок. Для этого нужно в соответствующий тэг кнопки добавить обработчик событий, реагирующий на щелчок мыши. Он называется onClick:

```
<INPUT TYPE="button" VALUE="Убрать фоновый рисунок" onClick="noBg
()">
```

Аналогично создадим функцию для смены цвета фона на белый:

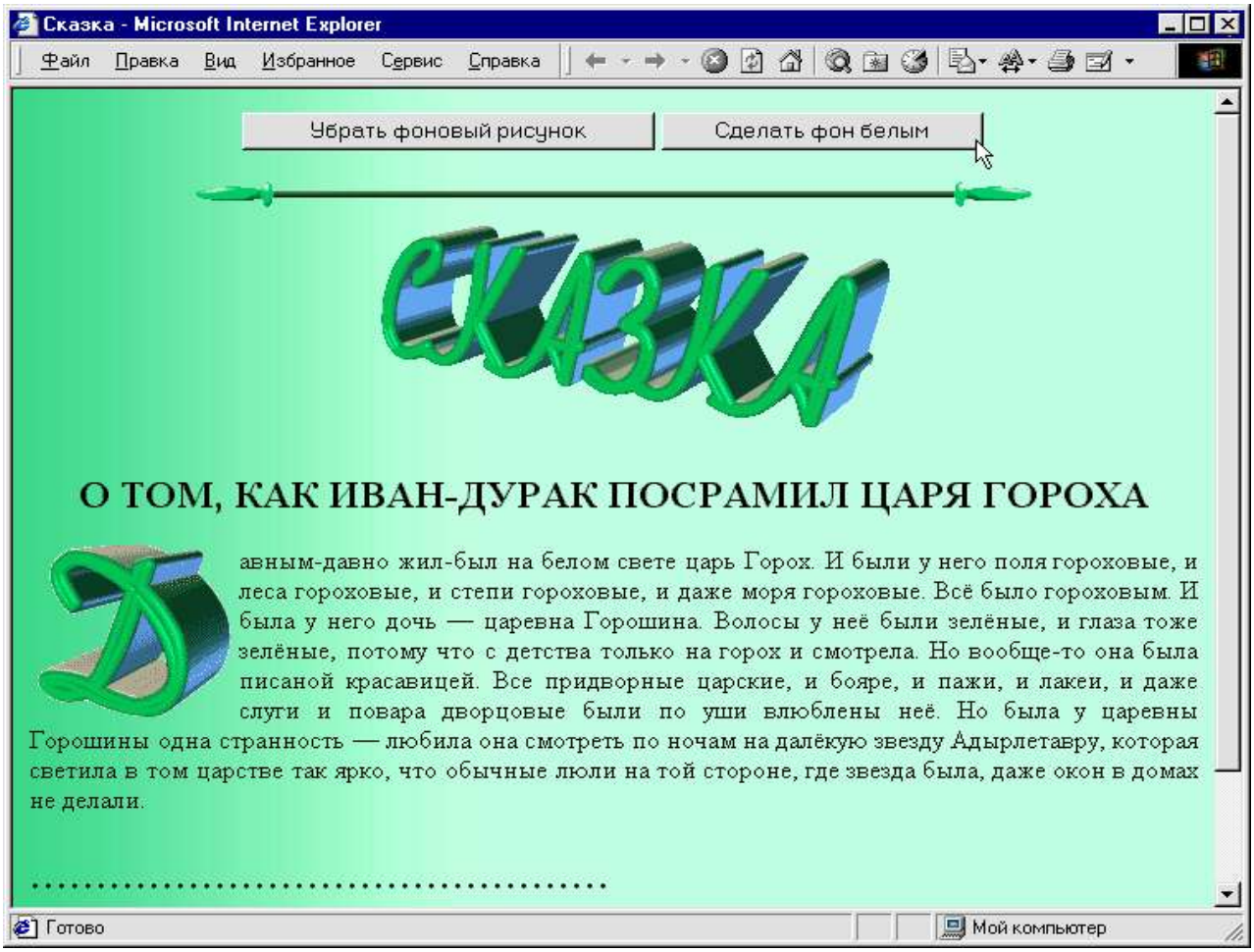
```
function colChange() {
     document.all.doc.style.backgroundColor='white';
}
```

и добавим ко второй кнопке обработчик события onClick:

```
<INPUT TYPE="button" VALUE="Сделать фон белым" onClick="colChange
()">
```

Если теперь открыть эту страницу в браузере (рис.7.6), то при нажатии на кнопку «Убрать фоновый рисунок» градиентный фоновый перелив исчезнет, уступив место зеленоватому, а при нажатиив на вторую кнопку «сделать фон белым» фон страницы действительно станет белым.

Однако неплохо бы дать пользователю и возможность обратной смены цвета и включения градиента (он, конечно, может для этого нажать в браузере кнопку «Обновить»), однако рассчитывать на это некорректно, да и не всякий пользователь сразу сообразит это сделать. Поэтому давайте, во первых, при выключении фонового рисунка сменим надпись на кнопке. Для этого желательно дать кнопкам имена, например, вот так:




```
<INPUT TYPE="button" NAME="butt1" VALUE="Убрать фоновый рисунок"
onClick="noBg()">
<INPUT TYPE="button" NAME="butt2" VALUE="Сделать фон белым"
onClick="colChange()">
```

Можно было, разумеется, использовать и атрибут ID вместо NAME. Для того, чтобы изменить надпись на первой кнопке, достаточно изменить значение её атрибуту VALUE:

```
document.all.butt1.value='Вернуть фоновый рисунок';
```

Теперь давайте подумаем, как нам переделать функцию noBg(). Ведь она должна убирать фоновый рисунок, если он есть, и включать его, если его нет. Одновременно нужно соответствующим образом изменять надпись на кнопке. Следовательно, нужно вначале проверить, есть ли фоновый рисунок:

```
function noBg() {
     if (document.all.doc.style.backgroundImage!="none") {
       document.all.doc.style.backgroundImage='none';
       document.all.butt1.value='Вернуть фоновый рисунок';
     }
     else {
```

```
 document.all.doc.style.backgroundImage="url('Images/grad1.jpg')";
       document.all.butt1.value='Убрать фоновый рисунок';
     }
}
```

В первой строке функции мы сравниваем значение стилевого свойства backgroungImage со значением none, и если оно не совпадает с ним, то, значит фоновый рисунок есть. В этом случае мы присваиваем этому свойству значение none и изменяем надпись на кнопке на «Вернуть фоновый рисунок». В противном же случае мы присваиваем свойству backgroungImage значение, содержащее имя файла фонового рисунка, и изменяем надпись на кнопке на первоначальную.[2]

Точно таким же способом мы переделываем функцию colChange(). Вначале проверим значение свойства backgroundColor, и если оно не совпадает со значением «white» (белый), присвоим ему это значение, а в противном случае присвоим ему первоначальное значение #BFFFBF. Одновременно будем изменять и надпись на второй кнопке:

```
function colChange() {
     if (document.all.doc.style.backgroundColor!='white') {
          document.all.doc.style.backgroundColor='white';
          document.all.butt2.value='Сделать фон зелёным';
     }
     else {
          document.all.doc.style.backgroundColor='#BFFFBF';
          document.all.butt2.value='Сделать фон белым';
     }
}
```

Теперь пользователь может с помощью первой кнопки включать и выключать фоновый рисунок по желанию, а с помощью второй — переключать цвет фона с зеленоватого на белый и обратно. Однако остался ещё один нерешённый вопрос. Дело в том, что если включён фоновый рисунок, то переключение цвета фона не даёт никакого видимого результата, что может смутить пользователя.

Поэтому, пока включён фоновый рисунок, лучше вообще не давать пользователю возможности нажимать на вторую кнопку. Internet Explorer позволяет сделать её недоступной с помощью установки атрибута DISABLED[3]. Поскольку изначально фоновый рисунок включён, установим этот атрибут сразу же при создании второй кнопки:

```
<INPUT TYPE="button" NAME="butt2" VALUE="Сделать фон белым"
DISABLED onClick="colChange()">
```

А в функцию noBg() следует включить сброс атрибута DISABLED для второй кнопки при выключении фонового рисунка и, соответственно, установку этого атрибута при включении фона:

```
function noBg() {
```

2 Интересно, что если написать в первой строке функции if (document.all.doc.style.backgroundImage=="url ('Images/grad1.jrg')"), то в Internet Explorer 5 ничего не получится! Дело в том, что изначально стилевое свойство background-image инициализируется с помощью таблицы стилей (CSS), а не JavaScript, поэтому JavaScript-функция считает, что его значением является пустая строка!

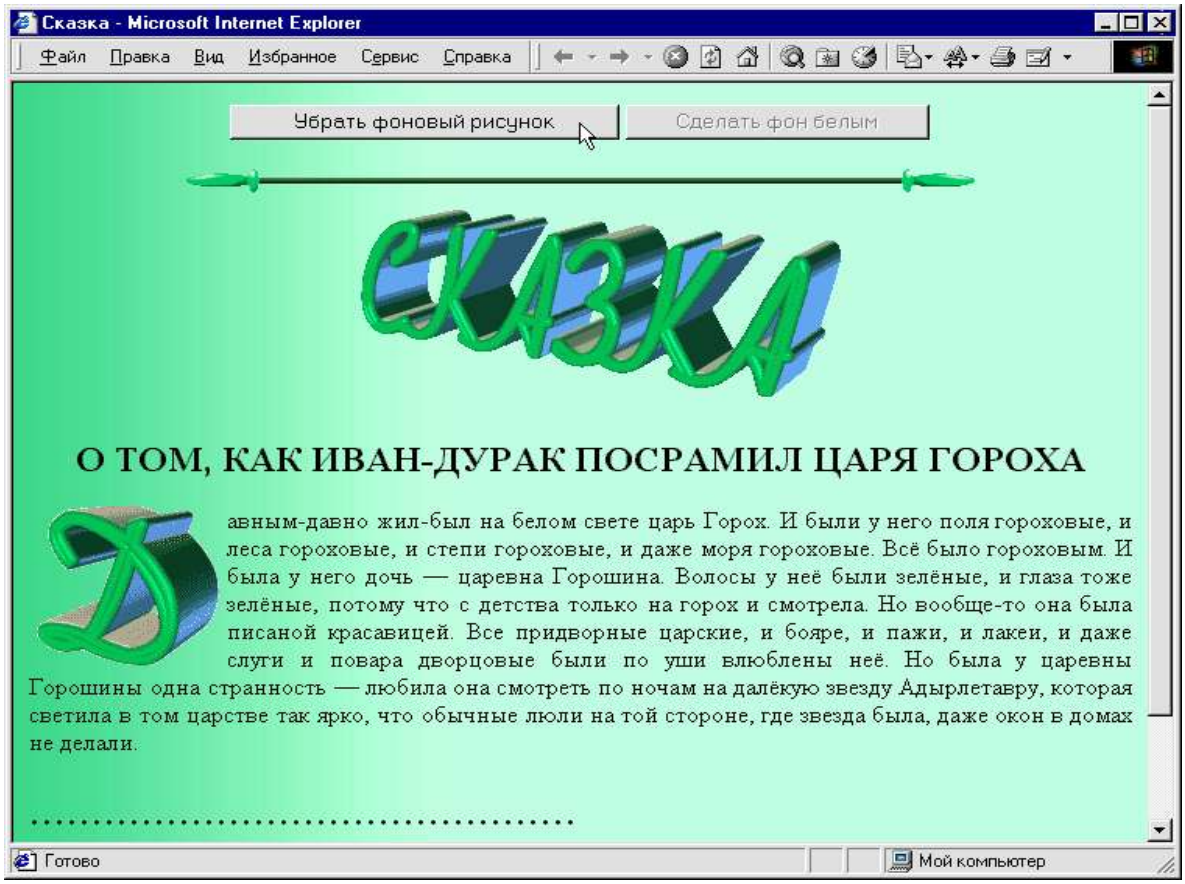
3 В Netscape 6 также есть эта возможность.

```
     if (document.all.doc.style.backgroundImage!="none") {
          document.all.doc.style.backgroundImage='none';
          document.all.butt1.value='Вернуть фоновый рисунок';
          document.all.butt2.disabled=false;
     }
     else {
 document.all.doc.style.backgroundImage="url('Images/grad1.jpg')";
          document.all.butt1.value='Убрать фоновый рисунок';
          document.all.butt2.disabled=true;
     }
}
```

Вот теперь всё становится на свои места. Пользователь сможет только тогда воспользоваться кнопкой для смены цвета фона, когда фоновый рисунок выключен. Можно было, конечно, и просто делать вторую кнопку невидимой (с помощью стилевого свойства visibility), однако, на наш взгляд, это было бы менее наглядно. Давайте посмотрим, что у нас получилось в целом:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 TRANSITIONAL//EN">
<HTML>
<HEAD>
     <TITLE>Сказка</TITLE>
<STYLE>
     BODY {    background-color: #BFFFBF;
               background-image: url('Images/grad1.jpg');
     }
</STYLE>
<SCRIPT>
<!--
function noBg() {
     if (document.all.doc.style.backgroundImage!="none") {
          document.all.doc.style.backgroundImage='none';
          document.all.butt1.value='Вернуть фоновый рисунок';
          document.all.butt2.disabled=false;
     }
     else {
 document.all.doc.style.backgroundImage="url('Images/grad1.jpg')";
          document.all.butt1.value='Убрать фоновый рисунок';
          document.all.butt2.disabled=true;
     }
}
function colChange() {
     if (document.all.doc.style.backgroundColor!='white') {
          document.all.doc.style.backgroundColor='white';
          document.all.butt2.value='Сделать фон зелёным';
     }
     else {
          document.all.doc.style.backgroundColor='#BFFFBF';
          document.all.butt2.value='Сделать фон белым';
     }
}
//-->
```

```
</SCRIPT>
</HEAD>
<BODY ID="doc">
<DIV ALIGN="center">
<INPUT TYPE="button" NAME="butt1" VALUE="Убрать фоновый рисунок"
onClick="noBg()">
<INPUT TYPE="button" NAME="butt2" VALUE="Сделать фон белым"
DISABLED onClick="colChange()">
</DIV><BR>
<DIV ALIGN="center"><IMG SRC="Images/hr2.gif" WIDTH="508"
HEIGHT="18" BORDER="0" ALT=""></DIV>
<DIV ALIGN="center">
<IMG SRC="Images/skazk.gif" WIDTH="359" HEIGHT="150" BORDER="0"
ALT="СКАЗКА"><BR>
<H2> О ТОМ, КАК ИВАН-ДУРАК ПОСРАМИЛ ЦАРЯ ГОРОХА</H2></DIV>
<DIV ALIGN="justify"><IMG SRC="Images/bukvica1.gif" WIDTH="121"
HEIGHT="111" BORDER="0" ALIGN="LEFT" ALT="Д">
авным-давно жил-был на белом свете царь Горох. И были у него поля
гороховые, и леса гороховые, и степи гороховые, и даже моря
гороховые. Всё было гороховым. И была у него дочь &mdash; царевна
Горошина. Волосы у неё были зелёные, и глаза тоже зелёные, потому
что с детства только на горох и смотрела. Но вообще-то она была
писаной красавицей. Все придворные царские, и бояре, и пажи, и
лакеи, и даже слуги и повара дворцовые были по уши влюблены неё.
Но была у царевны Горошины одна странность &mdash; любила она
смотреть по ночам на далёкую звезду Адырлетавру, которая светила в
том царстве так ярко, что обычные люди на той стороне, где звезда
была, даже окон в домах не делали.<BR><BR>
<FONT SIZE="+3">.....
</FONT><BR><BR>
Тут и сказке конец, а кто слушал &mdash; молодец, ему пряник в
награду и кило мармеладу.<BR> </DIV>
<DIV ALIGN="center"><IMG SRC="Images/hr2.gif" WIDTH="508"
HEIGHT="18" BORDER="0" ALT=""></DIV>
</BODY>
</HTML>
```



Результат показан на рис.7.7 (так страница будет выглядеть сразу после загрузки). Обратите внимание, что вторая кнопка изначально недоступна. Для красоты мы отделили наши кнопки от основного текста тем же разделителем, который использован в конце страницы.

Заметим, что приведённая выше страница будет работать только в Internet Explorer. Вы можете заставить работать её и в Netscape 6 тем же способом, который мы применяли в предыдущем примере — написания функции, присваивающей одной и той же переменной либо значение document.all, либо document.getElementById, в зависимости от типа браузера:

```
var doc, butt1, butt2;
function brws() {
     if (navigator.appName!="Netscape") {
          butt1=document.all.butt1;
          butt2=document.all.butt2;
          doc=document.all.doc;
          }
     else {
          butt1=document.getElementById('butt1');
          butt2=document.getElementById('butt2');
          doc=document.getElementById('doc');
          }
}
```

Затем следует переписать функции noBg() и colChange(), убрав из них обращение document.all:

```
function noBg() {
     if (doc.style.backgroundImage!="none") {
          doc.style.backgroundImage='none';
          butt1.value='Вернуть фоновый рисунок';
          butt2.disabled=false;
     }
     else {
          doc.style.backgroundImage="url('Images/grad1.jpg')";
          butt1.value='Убрать фоновый рисунок';
          butt2.disabled=true;
     }
}
function colChange() {
     if (doc.style.backgroundColor!='white') {
          doc.style.backgroundColor='white';
          butt2.value='Сделать фон зелёным';
     }
     else {
          doc.style.backgroundColor='#BFFFBF';
          butt2.value='Сделать фон белым';
     }
}
```

И, наконец, назначить выполнение функции brws() сразу после загрузки страницы:

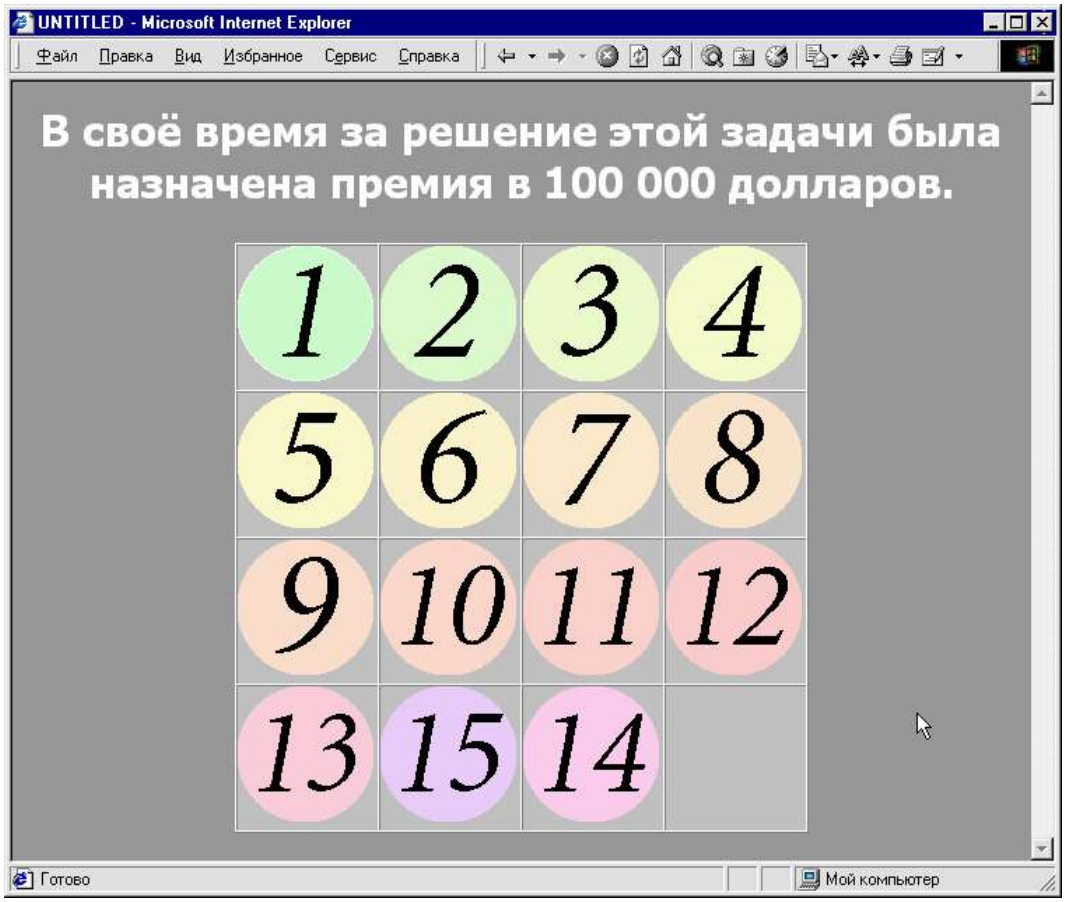
```
<BODY ID="doc" onLoad="brws()">
```

Теперь наш код будет одинаково работать и в Internet Explorer 4+[4], и в Netscape 6. Что касается Netscape 4, то если постараться, можно заставить эту страницу работать и там, но это будет довольно сложно (в Netscape 4 нельзя получить доступ непосредственно к свойствам BODY, однако можно «обмануть» его, использовав либо тэг <LAYER> и коллекцию document.layers, либо свойства типа document.body.bgColor). Давайте лучше не будем этим заниматься, тем более что смотреть нашу страницу в Netscape 4 всё равно можно, просто обе наши кнопки в нём вообще не отобразятся (этот браузер не воспринимает тэги <INPUT> вне элемента <FORM>).

Итак, мы с вами рассмотрели несколько основных обработчиков событий. Однако существуют и другие события мыши. Например, веб-страничка может отдельно реагировать на нажатие кнопки мыши, на её отпускание и даже на её движение. Для чего это может понадобиться?

Одно из возможных применений — это реализация так называемой технологии drag-n-drop, проще говоря — перетягивания экранных объектов с помощью мыши. Для иллюстрации того, как можно реализовать технологию drag-n-drop, рассмотрим несложный пример.

4 Так обозначается словосочетание "версии 4 и выше".



Предположим, вы хотите проиллюстрировать на своей страничке знаменитую игру Лойда «Пятнадцать», например, так, как показано на рис.7.8. Вообще говоря, если вы уже прочитали главы 2, 3 и 4 этой книги, то, скорее всего, сразу сообразите, как можно создать подобную страницу. Нужно, вроде бы, сначала просто задать стиль для текста:

```
<STYLE> BODY {
     background-color : #979797;
     color : #FEFEFE;
     text-align : center;
     font-weight : bold;
     font-size : 30px;
     font-family : sans-serif;
 }</STYLE>
```

затем вывести на экран заголовок; потом создать центрированный блок (<DIV>) с фиксированной шириной и высотой, а также небольшим отступом сверху, заданным с помощью стилевого свойства margin-top:

```
<DIV ALIGN="center" STYLE="width: 400px; height: 400px; margin-
top: 25px;">
```

Теперь осталось вставить в этот блок таблицу, у которой был бы определён фоновый цвет, отличающийся от основного фона страницы, а также тонкие границы между ячейками:

```
<TABLE BGCOLOR="#C0C0C0" WIDTH="100%" CELLSPACING="0"
```

```
CELLPADDING="0" BORDER="1">
```

и в каждую ячейку этой таблицы поместить заранее подготовленное изображение плашки с цифрой[5], например, вот так:

```
<TD WIDTH="25%" ID="c1"><IMG SRC="Images/digit1.gif" WIDTH="100"
HEIGHT="100" BORDER="0" ALT="1"></TD>
```

Вообще говоря, поскольку ширина (и высота) рисунков определены в 100 пикселей, атрибут WIDTH указывать для тэга <TD> совсем не обязательно. Что же касается атрибута ID, то мы здесь указали его с расчётом на то, что плашки придётся переставлять, и тогда потребуется доступ к ячейкам таблицы[6].

Всё это замечательно, если исходная позиция задана изначально. Однако предположим, что вы хотите дать пользователю возможность самому расставить плашки. Пусть вначале все они расположены вне игрового поля, можно даже расположить их «друг на друге». Пользователь при этом должен будет иметь возможность перетянуть мышью каждую из них на одну из «клеток» игрового поля.

Для этого придётся сделать три вещи. Во-первых, при нажатии пользователем кнопки мыши нужно определить, в каком месте окна браузера она нажата. Если нажатие произошло на рисунке плашки, нужно сразу же «привязать» этот рисунок к указателю мыши, чтобы он передвигался вместе с ним, пока кнопка не будет отпущена.

Во-вторых, во время движения указателя мыши необходимо передвигать вслед за ним этот рисунок (но только в том случае, если пользователь ещё не отпустил кнопку мыши). Если же мышь передвигается с отпущенной кнопкой, ничего происходить не должно.

И, в-третьих, при отпускании кнопки мыши нужно оставить рисунок на том месте, куда его передвинул пользователь.

Для того, чтобы всё это реализовать, придётся использовать позиционирование объектов на экране. Прежде всего, определим позицию блока, в который будет включена таблица («игровое поле»):

```
<DIV STYLE="width: 400px; height: 400px; position: absolute; top:
100px;">
```

Вероятно, вы обратили внимание, что сейчас мы задали только позицию блока <DIV> по вертикали (с помощью стилевого свойства top). А каким должно быть свойство left (позиция по горизонтали)? Хотелось бы, чтобы наше игровое поле располагалось по центру, но ведь мы не знаем ширину окна браузера пользователя!

К счастью, в Internet Explorer[7] есть возможность определить ширину окна браузера, прочитав значение свойства document.body.clientWidth. После этого всё уже просто. Сначала разделим это значение на 2, чтобы получить позицию в центре экрана. Поскольку наша таблица будет иметь ширину 400 пикселей, для вычисления позиции её левого края вычтем из позиции центра экрана половину ширины таблицы, то есть 200:

---

5 Поскольку плашки в нашем примере круглые (что сделано по эстетическим соображениям, однако не совсем соответствует привычной конструкции игры «Пятнадцать»), то, значит, стандартными средствами HTML их создать нельзя. Правда, можно это сделать в Internet Explorer с помощью элемента управления «Structured Graphics», о чём пойдёт речь в главе 9.

6 Мы не приводим здесь полностью исходный текст этой страницы, так как мы ещё вернёмся к ней в разделе 7.4.2

7 Обратите внимание, что описываемая веб-страница ориентирована только на Internet Explorer. Некоторые замечания по поводу её совместимости с Netscape 6 будут приведены ниже.

```
tstart=document.body.clientWidth/2-200;
```

В данном примере мы присвоили вычисленное значение переменной tstart. Поскольку это значение будет неоднократно использоваться в дальнейшем, полезно объявить эту переменную как глобальную (то есть не внутри какой-либо функции, а в самом начале кода JavaScript). Теперь осталось дать имя нашему блоку <DIV>:

```
<DIV ID="maintab" STYLE="width: 400px; height: 400px; position:
absolute; top: 100px;">
```

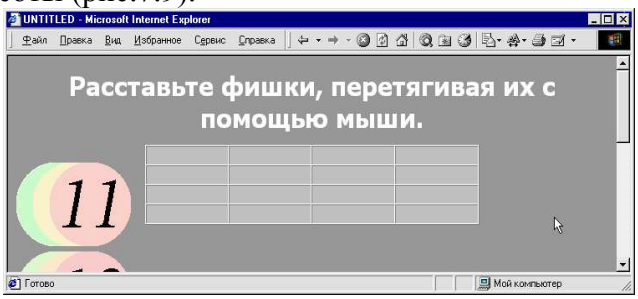
и присвоить его свойству left только вычисленное значение:

```
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var tstart;
function mainpos() {
     tstart=document.body.clientWidth/2-200;
     document.all.maintab.style.left=tstart;
}
//-->
</SCRIPT>
```

Приведённая выше функция mainpos() должна выполняться сразу после загрузки страницы (иначе пользователь увидит таблицу Бог знает где). Поэтому установим в тэге <BODY> уже знакомый нам обработчик событий onLoad:

```
<BODY onLoad="mainpos()">
```

Вот теперь при загрузке страницы (а точнее, сразу после неё) наш блок <DIV>, содержащий таблицу, будет отцентрирован. Но как быть с ячейками таблицы? Ведь мы знаем, что если в тэгах <TD> даже указать ширину и высоту, эти требования будут восприняты браузером лишь как рекомендательные, и на экране просто не отобразится ни одна ячейка. Если же поместить в ячейки неразрывные пробелы, то строки таблицы получатся мизерной высоты (рис.7.9).



Что же делать? Есть один приём, который позволяет сделать минимальную ширину и высоту ячейки таблицы такой, какой нужно именно вам, а не такой, какой захочет браузер. Дело в том, что при наличии рисунка в ячейке таблицы браузер обязательно расширит границы ячейки так, чтобы он был виден целиком. Поэтому создадим очень маленький графический файл, содержащий целиком прозрачный рисунок (в нашем примере был использован прозрачный рисунок размером 4×3 пикселя). Рисунок такого маленького

размера будет загружаться очень быстро, практически не влияя на скорость загрузки страницы. Однако в соответствующем ему тэге <IMG> установим ширину и высоту (с помощью атрибутов WIDTH и HEIGHT) такими, какими мы хотим видеть ширину и высоту ячейки таблицы. Таким образом, мы получим как бы пустую ячейку с заданными минимальными размерами[8]!

Вот как можно это сделать в нашем примере:

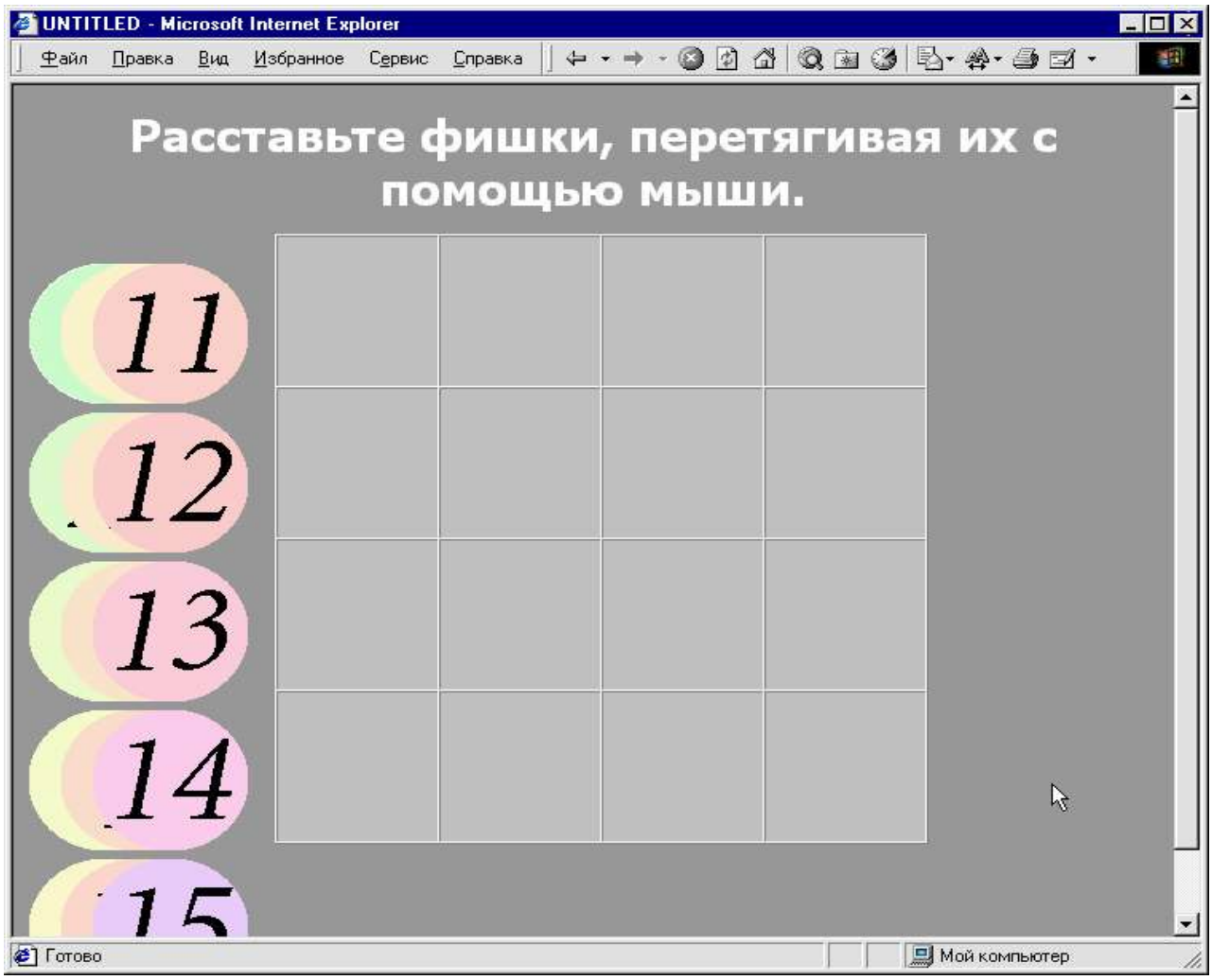
```
<TD><IMG SRC="Images/diafano1.gif" WIDTH="100" HEIGHT="100"></TD>
```

На всякий случай поместим этот рисунок в каждую ячейку таблицы. Теперь давайте поместим изображения плашек слева от игрового поля, расположив их вертикальными рядами по пять штук:

```
<IMG ID="p1" SRC="Images/digit1.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="1" STYLE="position: absolute; top: 120px; left:
10px;">
<IMG ID="p2" SRC="Images/digit2.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="2" STYLE="position: absolute; top: 220px; left:
10px;">
<IMG ID="p3" SRC="Images/digit3.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="3" STYLE="position: absolute; top: 320px; left:
10px;">
<IMG ID="p4" SRC="Images/digit4.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="4" STYLE="position: absolute; top: 420px; left:
10px;">
<IMG ID="p5" SRC="Images/digit5.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="5" STYLE="position: absolute; top: 520px; left:
10px;">
<IMG ID="p6" SRC="Images/digit6.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="6" STYLE="position: absolute; top: 120px; left:
30px;">
<IMG ID="p7" SRC="Images/digit7.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="7" STYLE="position: absolute; top: 220px; left:
30px;">
<IMG ID="p8" SRC="Images/digit8.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="8" STYLE="position: absolute; top: 320px; left:
30px;">
<IMG ID="p9" SRC="Images/digit9.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="9" STYLE="position: absolute; top: 420px; left:
30px;">
<IMG ID="p10" SRC="Images/digit10.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="10" STYLE="position: absolute; top: 520px; left:
30px;">
<IMG ID="p11" SRC="Images/digit11.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="11" STYLE="position: absolute; top: 120px; left:
50px;">
<IMG ID="p12" SRC="Images/digit12.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="12" STYLE="position: absolute; top: 220px; left:
50px;">
```

8 Рекомендуем использовать этот приём всегда, когда нужно сделать ячейку таблицы шире или выше, чем это «нужно» для вывода имеющегося в ней текста.

```
<IMG ID="p13" SRC="Images/digit13.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="13" STYLE="position: absolute; top: 320px; left:
50px;">
<IMG ID="p14" SRC="Images/digit14.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="14" STYLE="position: absolute; top: 420px; left:
50px;">
<IMG ID="p15" SRC="Images/digit15.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="15" STYLE="position: absolute; top: 520px; left:
50px;">
```

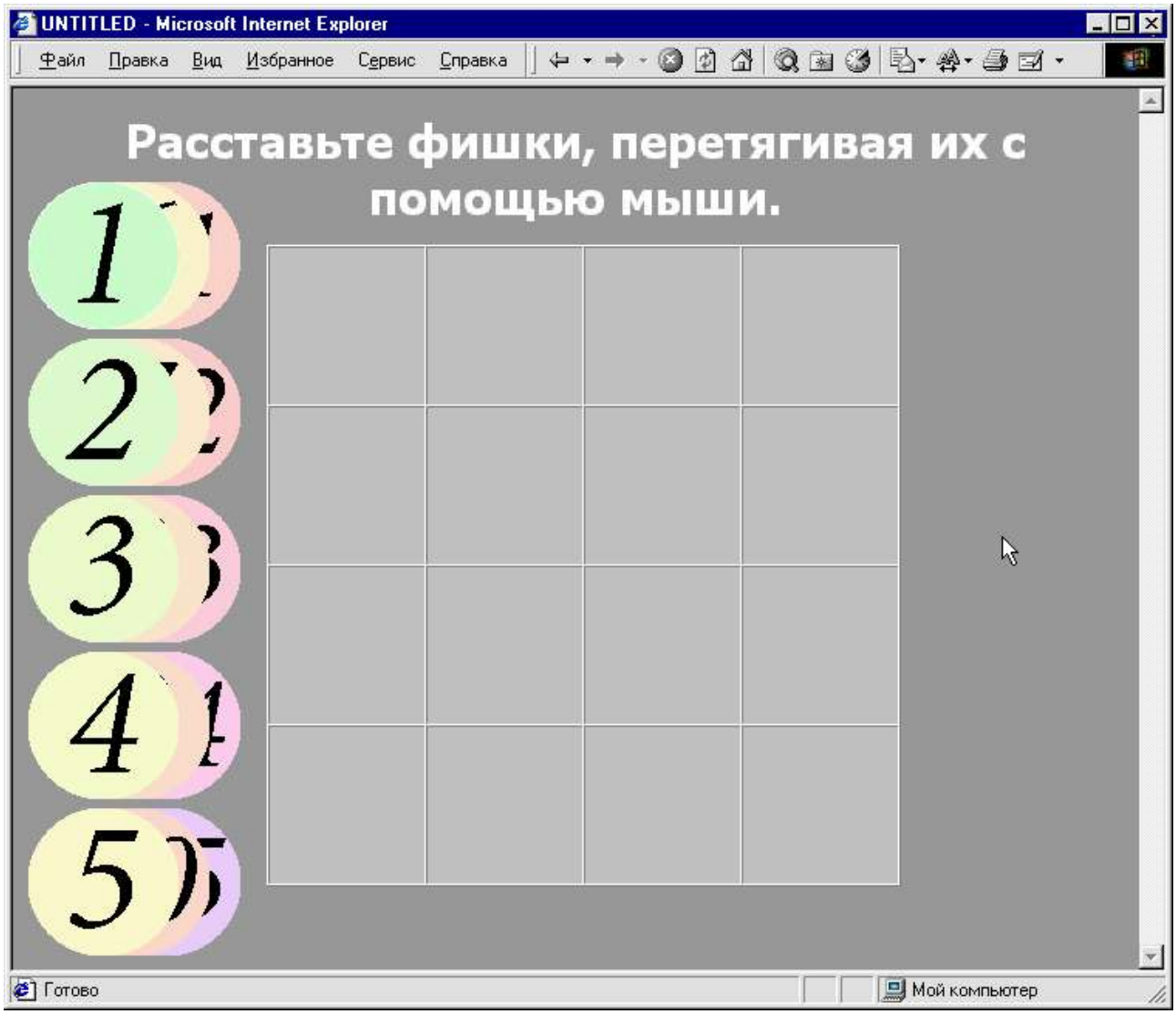


Результат показан на рис.7.10. Пока что это выглядит не очень красиво, поскольку на виду у пользователя оказался третий ряд плашек (с цифрами 11 по 15), а остальные расположились «под ним». Лучше было бы, если бы «наверху» оказались плашки с 1 по 5.

Можно, конечно, решить эту проблему, присвоив каждой плашке своё значение стилевого свойства z-index, однако проще изменить порядок следования тэгов <IMG>. Если сначала написать тэги для рисунков плашек с 11 по 15, затем — с 6 по 10 и в конце — с 1 по 5, то при наложении рисунков те, которые были объявлены позже, окажутся «сверху». Кроме того, вполне можно разместить рисунки чуть выше по вертикали:

```
<IMG ID="p11" SRC="Images/digit11.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="11" STYLE="position: absolute; top: 60px; left:
```

```
50px;">
<IMG ID="p12" SRC="Images/digit12.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="12" STYLE="position: absolute; top: 160px; left:
50px;">
<IMG ID="p13" SRC="Images/digit13.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="13" STYLE="position: absolute; top: 260px; left:
50px;">
<IMG ID="p14" SRC="Images/digit14.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="14" STYLE="position: absolute; top: 360px; left:
50px;">
<IMG ID="p15" SRC="Images/digit15.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="15" STYLE="position: absolute; top: 460px; left:
50px;">
<IMG ID="p6" SRC="Images/digit6.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="6" STYLE="position: absolute; top: 60px; left:
30px;">
<IMG ID="p7" SRC="Images/digit7.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="7" STYLE="position: absolute; top: 160px; left:
30px;">
… (и т.д.)
```



Результат показан на рис.7.11. Теперь, наконец, подготовительная работа закончена, и нам нужно реализовать обработку событий, как говорилось выше.

Сначала добавим в тэг <BODY> обработчики событий, реагирующие соответственно на нажатие кнопки мыши (не на щелчок, который состоит из нажатия и отпускания левой кнопки, а именно на нажатие) — onMouseDown, на отпускание кнопки — onMouseUp и на движение указателя мыши — onMouseMove:

```
<BODY onLoad="mainpos()" onMouseDown="down_it()" onMouseUp="up_it
()" onMouseMove="move_it()">
```

Теперь осталось написать функции, которые мы так лихо назначили обработчикам событий. Сначала давайте займёмся нажатием кнопки мыши (onMouseDown).

Прежде всего нам надо определить, была ли нажата кнопка мыши на рисунке одной из плашек. Если нет, то ничего делать не нужно.

Как проверить это условие? В Internet Explorer 4+ источник каждого события записывается в свойство window.event.srcElement[9]. Но что нам это даёт? Ведь нужных нам рисунков целых 15, и у каждого есть своё уникальное имя (свойство ID). Неужели придётся сравнивать window.event.srcElement.id с каждым именем?

Подождите, ведь мы совсем забыли, что у каждый рисунок плашки представляет собой тэг <IMG>. Поэтому мы можем сравнить свойство window.event.srcElement.tagName, содержащее названия тэга-источника события, со словом IMG, и в случае удачи перейти к дальнейшим действиям:

```
function down_it() {
     if(window.event.srcElement.tagName=="IMG") {
          // какие-то действия
      }
}
```

Стоп! Но ведь, кроме рисунков плашек, у нас ещё есть прозрачный рисунок, расположенный в каждой из 16 ячеек таблицы! А вот его-то нам никуда передвигать совсем не нужно. При этом от рисунков плашек его отличает только свойство SRC. Придётся сравнить это свойство со значением Images/diafano1.gif, и продолжать дальнейшую работу функции лишь в том случае, если совпадение не обнаружится.

Однако если мы напишем[10]

```
if((window.event.srcElement.tagName=="IMG")&&
(window.event.srcElement.src.!="Images/diafano1.gif")) {
          // какие-то действия
 }
```

то в большинстве случаев нас постигнет разочарование: «какие-то действия» всё равно будут выполняться, даже если кнопка мыши будет нажата на прозрачном рисунке! В чём же дело?

Оказывается, свойство window.event.srcElement.src в любом случае содержит абсолютное расположение рисунка (например, если мы делаем эти опыты на локальном компьютере, то значением window.event.srcElement.src будет полный путь к файлу, включающий имя диска и родительские папки). Поскольку при разработке странички вы, скорее всего, ещё не можете точно предсказать его будущее абсолютное расположение (да и проверить работу странички на локальном компьютере тоже нелишне), придётся поступить по другому. Воспользуемся тем, что каково бы ни было абсолютное расположение файла, значение window.event.srcElement.src всё равно будет заканчиваться именем файла, то есть в нашем случае diafano1.gif, то есть символы с 13-го по 5-й от конца строки будут заведомо содержать значение «diafano1». Поскольку длина любой строки всегда содержится в её свойстве length, то мы можем выделить из полного названия файла нужные нам символы, начиная от length–12 и кончая length–4. Выделение части строки можно произвести методом substring:

```
if((window.event.srcElement.tagName=="IMG")&&
(window.event.srcElement.src.substring
(window.event.srcElement.src.length-
12,window.event.srcElement.src.length-4)!="diafano1")) {
```

9 Вообще говоря, слово window можно опустить, так как объект window является определённым по умолчанию.

10 Напомним, что && в JavaScript означает логическую операцию «И».

```
            // какие-то действия
        }
```

Вот теперь всё заработает правильно. Правда, строка условия выглядит уж очень громоздко. Мы не особенно рекомендуем писать такие строки, поскольку через какое-то время вам самим будет трудновато в них разобраться, если вы вдруг захотите что-либо изменить. Например, в нашем случае можно определить локальную переменную l и присвоить ей значение window.event.srcElement.src.length. Тогда строка условия будет выглядеть хоть немного компактней:

```
if((window.event.srcElement.tagName=="IMG")&&
(window.event.srcElement.src.substring(l-12,l-4)!="diafano1"))
```

Какие же действия нужно осуществить внутри этой функции? Если вы ещё не забыли, нам нужно «привязать» рисунок плашки к указателю мыши. Для этого достаточно определить глобальную переменную (мы назвали её moving), и присваивать ей всегда имя рисунка, который необходимо передвигать. Если никакой рисунок передвигать не нужно (кнопка мыши отпущена или нажата не на рисунке плашки), можно присвоить переменной moving значение "" (пустая строка). В самом начале скрипта эту переменную можно объявить так:

```
var moving="";
```

а в теле функции down_it(), которую мы сейчас пишем, будем присваивать ей значение, содержащее имя того рисунка, на котором щёлкнул мышью пользователь:

```
moving=window.event.srcElement.id;
```

В принципе, наша функция down_it() уже справляется со своими «обязанностями». Теперь давайте займёмся функцией move_it(), которая будет вызываться при движении мыши.

Эта функция должна прежде всего проверить, нужно ли передвигать какой-либо рисунок. Как вы помните, его имя содержится в переменной moving. Так что нужно вначале сравнить значение переменной moving с пустой строкой и в случае совпадения не предпринимать никаких действий:

```
function move_it() {
     if (moving!="") {
               // какие-то действия
          }
}
```

Теперь давайте подумаем, что должно быть сделано, если переменная moving содержит имя рисунка, который нужно передвинуть. Очевидно, для того, чтобы его передвинуть, нужно изменить его стилевые свойства left и top в соответствии с расположением указателя мыши. Текущее положение указателя мыши можно узнать, прочитав значения свойств window.event.clientX и window.event.clientY.

Стоп, скажете вы. А как узнать, как должен располагаться рисунок *относительно указателя мыши*? Ведь пользователь может щёлкнуть и в центре рисунка, и с краю, и в любом другом месте. Значит, в функции down_it(), которую мы считали уже законченной, нужно ещё вычислить координаты указателя мыши относительно рисунка?

Правильно! Это обязательно нужно сделать, если вы будете применять эту технологию для перетягивания крупных объектов. Но в нашем примере мы позволим себе упростить себе задачу, воспользовавшись тем, что наши плашки имеют относительно небольшие размеры. При таких размерах будет вполне нормально смотреться, если при перетягивании рисунка указатель мыши будет находиться посередине его.

Поскольку рисунки наши имеют размер 100×100, нам остаётся вычесть 50 из каждой координаты указателя мыши и присвоить эти значения свойствам left и top рисунка:

```
document.all[moving].style.pixelLeft=window.event.clientX-50;
document.all[moving].style.pixelTop=window.event.clientY-50;
```

Обратите внимание: для того чтобы обратиться к объекту по его имени, содержащемся в переменной, необходимо использовать квадратные скобки, то есть писать document.all[moving], а не document.all.moving. В противном случае браузер не сможет найти нужный объект, и на экран вылезет сообщение об ошибке[11]. Кроме того, обратите внимание на то, что для корректного изменения координат в Internet Explorer необходимо использовать свойства pixelLeft и pixelTop вместо left и top.

В эстетических целях давайте передвинем центр рисунка к указателю мыши уже в функции down_it(), добавив туда две точно такие же строки. Что касается функции move_it(), то она почти готова. Однако необходимо добавить в неё ещё две строки, чтобы предотвратить заранее предопределённую реакцию браузера на какие-либо ситуации:

```
window.event.cancelBubble = true;
window.event.returnValue = false;
```

Первая из этих строк отменяет так называемое «всплытие» события, то есть возникновение его в элементах страницы, содержащих элемент-источник[12] А вторая строка запрещает браузеру выполнять действия, назначенные по умолчанию для этого события. В данном случае, если мы не напишем

```
window.event.returnValue = false;
```

то рисунки начнут «тормозиться» уже при небольшом перемещении, после чего браузер может вообще не распознать отпускание кнопки мыши. Мы сейчас не будем вдаваться в подробности того, почему так происходит. Однако запомните, что эти две короткие строчки при обработке событий мыши часто помогают избежать многих неприятностей. Если у вас что-то не получается, проверьте, а не «всплывает» ли какое-нибудь нежелательное событие? И не пытается ли браузер делать что-то «своё» вместо назначенных вами операций?

---

11 Другим способом обратиться к объекту через переменную является использование встроенной функции eval (), например, в данном случае можно написать eval('document.all'+moving+'stye.pixelLeft').

12 Например, в данном случае при отсутствии этой строки событие onMouseDown, возникшее на рисунке (<IMG>), будет передано также элементу <BODY>, который содержит этот рисунок. В нашей ситуации, правда, ничего страшного при этом не произойдёт, однако на всякий случай лучше «обезопаситься«.

Теперь давайте займёмся функцией up_it(), выполняющейся при отпускании кнопки мыши. Собственно говоря, всё, что нужно сделать — это проверить, передвигался ли какой-нибудь объект (то есть содержит ли переменная moving какое-либо имя) и, если это так, присвоить этой переменной пустую строку, что будет означать «освобождение» рисунка:

```
function up_it() {
     if (moving!="") moving="";
}
```

Однако хорошо бы ещё расположить рисунок не где попало, а точно в ячейке таблицы. Поскольку в этом случае его координаты относительно начала таблицы должны быть кратны 100, это довольно легко осуществить. Для этого достаточно округлить его до ближайшей сотни. Для округления можно использовать встроенный метод Math.round. Понятно, что он округляет не до сотен, а до целых чисел, поэтому текущие координаты рисунка перед округлением придётся разделить на 100, а после округления — умножить на 100. Кроме того, не забывайте, что кратность 100 мы определяем относительно начала таблицы, которое равно tstart по горизонтали и 100 по вертикали. Поэтому перед делением на 100 нужно ещё вычесть из горизонтальной координаты значение tstart, а в конце снова его прибавить. Вот что у нас получается:

```
document.all[moving].style.pixelLeft=Math.round
((window.event.clientX-50-tstart)/100)*100+tstart+1;
document.all[moving].style.pixelTop=Math.round
((window.event.clientY-50)/100)*100+1;
```

Как видите, всё довольно просто. Здесь мы прибавили к каждой координате ещё по единице, чтобы рисунки не «налезали» на сетку таблицы. Кстати, ширину ячеек таблицы (то есть прозрачного рисунка diafano1.gif) в этом случае тоже необходимо немного скорректировать. Поскольку каждая ячейка таблицы имеет со всех сторон «бордюр» шириной в 1 пиксель, придётся сделать ширину самих ячеек равной не 100, а 98:

```
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
```

Кроме того, неплохо было бы, если бы наши рисунки располагались точно по сетке таблицы только в её пределах, а в других частях экрана принимали бы свободное положение. Для этого можно перед выполнением «округления до сотен» проверить, расположен ли рисунок внутри таблицы (или хотя бы рядом с ней)[13]:

```
if (window.event.clientX>=tstart-50&&window.event.clientY>=50) {
document.all[moving].style.pixelLeft=Math.round
((window.event.clientX-50-tstart)/100)*100+tstart+1;
document.all[moving].style.pixelTop=Math.round
((window.event.clientY-50)/100)*100+1;
}
```

И, наконец, ещё один штрих. При перемещении некоторых рисунков может возникнуть ситуация, когда перемещаемый рисунок

13 В данном примере производится проверка только на нахождение рисунка левее или выше таблицы. Если рисунок находится правее или ниже её, то всё равно производится округление координат. Для упражнения вы можете сами модифицировать эти строки так, чтобы при нахождении рисунка правее или ниже таблицы округления координат не происходило.

будет проходить как бы под другим, пропадая на время из видимости. Чтобы этого не возникало, давайте добавим в функцию down_it() ещё такую строку:

```
document.all[moving].style.zIndex=5;
```

Поскольку у всех остальных элементов значение z-index не изменялось (и, следовательно, равно 0), мы добиваемся того, что перемещаемый рисунок никогда не будет перекрыт другими объектами. Естественно, при окончании перемещения рисунка ему нужно возвратить исходное значение z-index. Для этого в функцию up_it() добавим строку

document.all[moving].style.zIndex=0;

Итак, давайте посмотрим, что же у нас получается:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
     <TITLE>Игра 15</TITLE>
<STYLE> BODY {
     background-color : #979797;
     color : #FEFEFE;
     text-align : center;
     font-weight : bold;
     font-size : 30px;
     font-family : sans-serif;
 }
</STYLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
var tstart;
var moving="";
function mainpos() {
     tstart=document.body.clientWidth/2-200;
     document.all.maintab.style.left=tstart;
}
function down_it() {
     var l=window.event.srcElement.src.length;
     if((window.event.srcElement.tagName=="IMG")&&
(window.event.srcElement.src.substring(l-12,l-4)!="diafano1")) {
     moving=window.event.srcElement.id;
     document.all[moving].style.pixelLeft=window.event.clientX-50;
     document.all[moving].style.pixelTop=window.event.clientY-50;
     document.all[moving].style.zIndex=5;
      }
}
function up_it() {
     if (moving!="") {
  if(window.event.clientX>=tstart-50&&window.event.clientY>=50) {
document.all[moving].style.pixelLeft=Math.round
```

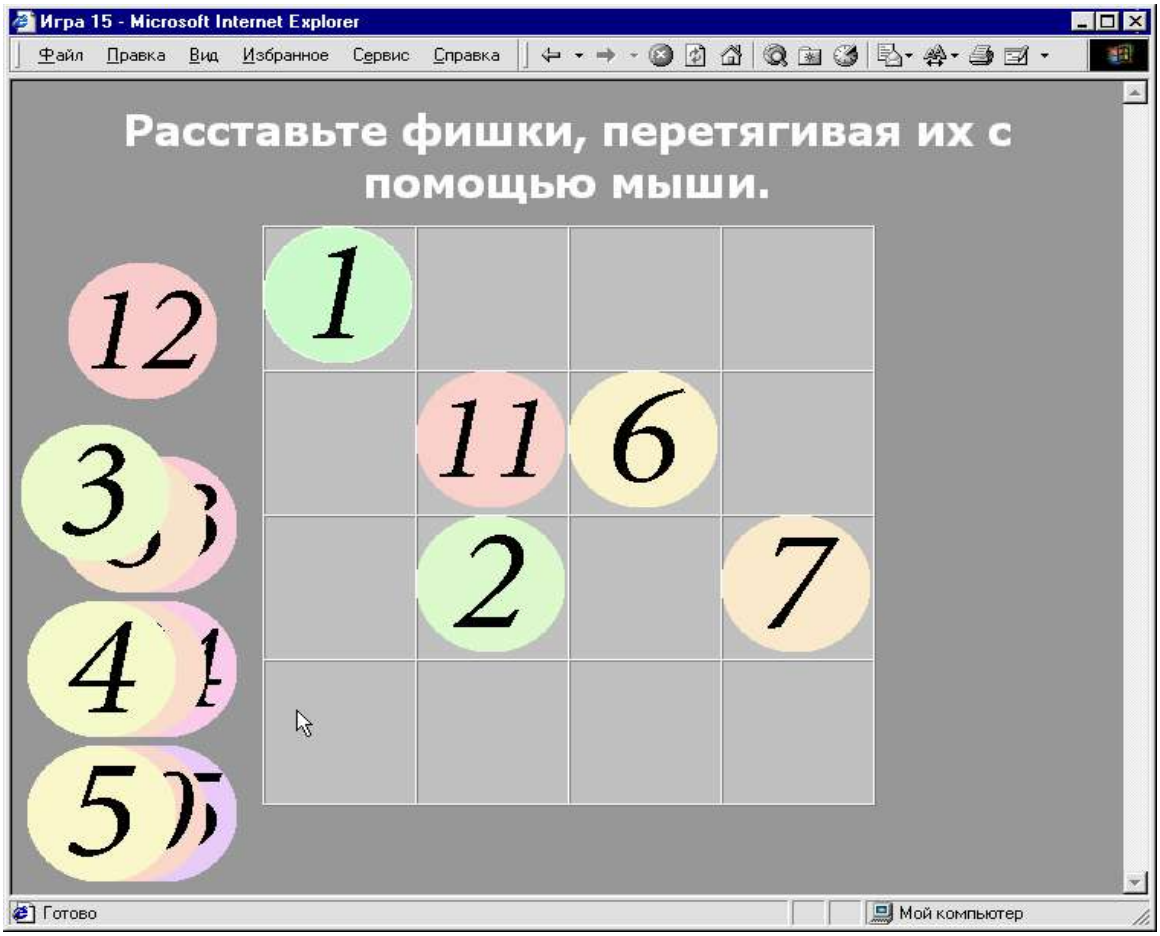
```
((window.event.clientX-50-tstart)/100)*100+tstart+1;
document.all[moving].style.pixelTop=Math.round
((window.event.clientY-50)/100)*100+1;
               }
          document.all[moving].style.zIndex=0;
          moving="";
     }
}
function move_it() {
     if (moving!="") {
      document.all[moving].style.pixelLeft=window.event.clientX-50;
      document.all[moving].style.pixelTop=window.event.clientY-50;
          }
      event.cancelBubble = true;
      event.returnValue = false;
}
//-->
</SCRIPT>
</HEAD>

<BODY onLoad="mainpos()" onMouseDown="down_it()" onMouseUp="up_it
()" onMouseMove="move_it()">

Расставьте фишки, перетягивая их с помощью мыши.
<DIV ID="maintab" STYLE="width: 400px; height: 400px; position:
absolute; top: 100px;">
<TABLE BGCOLOR="#C0C0C0" WIDTH="100%" CELLSPACING="0"
CELLPADDING="0" BORDER="1">
<TR>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
</TR>
<TR>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
</TR>
<TR>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
</TR>
<TR>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
<TD><IMG SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>
</TR>
```

```
</TABLE>
</DIV>

<IMG ID="p11" SRC="Images/digit11.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="11" STYLE="position: absolute; top: 60px; left:
50px;">
<IMG ID="p12" SRC="Images/digit12.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="12" STYLE="position: absolute; top: 160px; left:
50px;">
<IMG ID="p13" SRC="Images/digit13.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="13" STYLE="position: absolute; top: 260px; left:
50px;">
<IMG ID="p14" SRC="Images/digit14.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="14" STYLE="position: absolute; top: 360px; left:
50px;">
<IMG ID="p15" SRC="Images/digit15.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="15" STYLE="position: absolute; top: 460px; left:
50px;">
<IMG ID="p6" SRC="Images/digit6.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="6" STYLE="position: absolute; top: 60px; left:
30px;">
<IMG ID="p7" SRC="Images/digit7.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="7" STYLE="position: absolute; top: 160px; left:
30px;">
<IMG ID="p8" SRC="Images/digit8.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="8" STYLE="position: absolute; top: 260px; left:
30px;">
<IMG ID="p9" SRC="Images/digit9.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="9" STYLE="position: absolute; top: 360px; left:
30px;">
<IMG ID="p10" SRC="Images/digit10.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="10" STYLE="position: absolute; top: 460px; left:
30px;">
<IMG ID="p1" SRC="Images/digit1.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="1" STYLE="position: absolute; top: 60px; left:
10px;">
<IMG ID="p2" SRC="Images/digit2.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="2" STYLE="position: absolute; top: 160px; left:
10px;">
<IMG ID="p3" SRC="Images/digit3.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="3" STYLE="position: absolute; top: 260px; left:
10px;">
<IMG ID="p4" SRC="Images/digit4.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="4" STYLE="position: absolute; top: 360px; left:
10px;">
<IMG ID="p5" SRC="Images/digit5.gif" WIDTH="100" HEIGHT="100"
BORDER="0" ALT="5" STYLE="position: absolute; top: 460px; left:
10px;">
</BODY>
</HTML>
```



Результат вы можете видеть на рис.7.12. В принципе, в такую «игру» уже можно по-настоящему играть. Конечно, вы можете сказать, что этот код можно немного упростить. Правильно, зачем 16 раз повторять тэг вставки прозрачного рисунка? Давайте заменим его вложенным циклом JavaScript:

```
for (var k=1; k<=4; k++) {
     document.write('<TR>');
     for (var m=1; m<=4; m++) document.write('<TD><IMG
SRC="Images/diafano1.gif" WIDTH="98" HEIGHT="98"></TD>');
     document.write('</TR>');
}
```

Результат будет тот же. А если немного подумать, то можно сократить даже код первоначального расположения рисунков плашек, правда, это немного труднее.

Кроме того, пока что мы никак не проверяем, не ставит ли пользователь две плашки в одну и ту же ячейку, а уж о самой игре и говорить нечего. Но ведь мы пока только реализовывали расстановку плашек методом drag-n-drop. А что касается реализации игры, то мы к ней ещё вернёмся, когда будем говорить о динамической смене графических изображений. А пока что запомните рассмотренные в этом разделе приёмы, так как они позволяют организовать столь любимую пользователями интерактивность просто на небывалом уровне — вспомните, что одна из функций реагировала у нас буквально на каждое перемещение указателя мыши!

К сожалению, приведённая выше страница будет работать только в Internet Explorer. Если вы хотите, чтобы она работала также и в Netscape 6, вам придётся приложить некоторые усилия. Дело в том, что помимо различий в синтаксисе доступа к элементам, о котором мы уже говорили (в Netscape используется конструкция document.getElementById вместо document.all), различия существуют также и в обработке событий. В частности, вместо глобального объекта event в Netscape необходимо использовать временную переменную, которой будет передаваться значение объекта event. Кроме того, вместо свойства srcElement используется свойство target, а свойство returnValue вообще не поддерживается. Выше мы приводили примеры того, как написать код, работающий в обоих популярных браузерах. Вы можете в качестве упражнения попробовать это сделать и для данного примера, однако из-за обработки событий мыши это будет сложнее, чем в предыдущих случаях.