МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по программированию web-приложений

Учебно-методическое пособие

Часть 1. Введение в JavaScript

2018 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по программированию web-приложений. Учебно-методическое пособие. Часть 1. Введение в JavaScript. Озерск: ОТИ НИЯУ МИФИ, 2018. — 54 с.

В пособии описываются задания практических работ по дисциплине «Современные технологии программирования». Работы третьего семестра изучения дисциплины включают в себя основы программирования web-приложений с использованием языков программирования JavaScript, PHP, XPATH, XSLT и объектной модели документа DOM.

В качестве основного материала при выполнении практических работ пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника» и специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Общие цели занятий

В ходе практических работ изучается использование языков программирования JavaScript и PHP для формирования интерактивных web-страниц и web-приложений в целом.

Целью работ является создание приложения, моделирующего сетевую игру под названием «Королевский квадрат». Весь процесс создания этого приложения условно разбит на три логические части.

При этом первая часть работ в большей степени посвящена разработке алгоритмов игры на языке JavaScript в виде автономной страницы HTML, которая представляет собой законченное приложение.

Во второй части приложение переносится на сервер, и изучается взаимодействие серверной и клиентской частей кода с использованием языка PHP, а также технологий, связанных с представлением и обработкой информации, представленной в виде документа XML.

В третьей части основной задачей является исследование принципов интерактивной связи сервера с клиентом.

На выполнение каждой работы предположительно отводится от 2-х до 4-х академических часов. Успешное усвоение изучаемого материала возможно при условии, что студент самостоятельно находит необходимую техническую информацию, используя сеть Интернет, конспекты лекций и дополнительную литературу.

1. Работа JS-01. Введение в HTML и CSS (2 часа)

Цели:
- изучение структуры web-страницы;
- изучение тегов языка разметки HTML.
Задачи:
- управление оформлением содержания;
- управление размещением содержания.

1.1. Теги HTML

Для формирования web-страниц применяется язык разметки гипертекста HTML. В его основе лежит использование *тегов* (*tag*), которые выделяют часть содержания страницы для тех или иных целей. Тегом является одно из зарезервированных слов, заключенное в треугольные скобки. Например, тег html начинает описание страницы HTML (web-страницы).

Различают закрываемые и не закрываемые теги.

Закрываемые теги обозначают часть содержания, при этом в начале обозначаемой части размещается открывающий тег, а в конце этой части размещается закрывающий тег. Перед зарезервированным словом закрывающего тега ставят знак «слеш». Например, описание содержания страницы HTML должно быть заключено в тег html следующим образом:

<html>*Описание содержания страницы*</html>

Не закрываемые теги описывают часть содержания непосредственно в теге при помощи *параметров* тега.

Параметр тега, — это конструкция вида *параметр*="*значение*".

Здесь параметр, — это одно из допустимых для данного тега слов, а значение, — его желаемое значение. Обратим внимание, что значение параметра *желательно* заключать в кавычки, двойные или одинарные.

Примером не закрываемого тега является тег img, вставляющий в содержание картинку: . Параметром является слово src (от *source*), а значением, — путь к картинке. Другой не закрываемый тег, — разрыв строки
, — не имеет никаких параметров.

Параметры могут присутствовать в любых тегах. Например, следующие параметры задают размеры элемента таблицы:

<td width="20px" height="20px"></td>.

Обычно содержание страницы выводится и размещается в окне браузера последовательно по мере чтения страницы. В зависимости от размеров окна браузера внешний вид страницы получается различным, одни и те же элементы страницы в разных случаях могут оказываться в разных местах. Для управления правильным размещением содержания на странице используются контейнеры. Наиболее распространенные контейнеры — это блок div и таблица table.

1.2. CSS

Оформление текста и других элементов содержания управляется каскадными таблицами стилей CSS. Есть три способа задать оформление.

В качестве примера рассмотрим задание красного цвета части текста.

Способ 1. Задать оформление, используя параметр тега.

```
<html>
<body>
  Это <span style='color: red'>красный</span>
</body>
</html>
```

Здесь слово «красный» отображается красным цветом потому, что оно заключено в тег span, параметр style которого задает цвет при помощи параметра color.

Способ 2. Оформление задается в таблице стилей, которая расположена непосредственно в странице, в ее заголовочной части.

```
<html>
<head>
<style>
 .red { color: red; }
</style>
</head>
<body>
 Это <span class='red'>красный</span>
</body>
</html>
```

Таблица стилей расположена в теге style. В таблице задан класс оформления red, указывающий красный цвет шрифта. Параметр class тега span указывает, что к тексту тега применяется класс red. Достоинство способа заключается в том, что оформление класса может быть применено неоднократно к разным элементам страницы, и изменение оформления в таблице стилей изменит внешний вид всех элементов.

Способ 3. Таблица стилей выносится из файла страницы в отдельный файл, имеющий расширение .css. Теперь одни и те же стили могут быть применены сразу ко многим страницам. Включение таблицы стилей в страницу выполняет тег link:

```
<html>
<head>
<link rel='stylesheet' type='text/css' href='page.css'/>
</head>
```

Таблица стилей находится в файле page.css, как указано в теге link.

Если на странице одновременно используются все три способа задания оформления, нужно помнить, что способ 1 имеет преимущество перед другими способами, а способ 2 имеет преимущество перед способом 3.

1.3. Цель работы

Конечной целью работ является создание web-приложения «Сетевая игра «Королевский квадрат». На рисунке 1 приведены основные элементы, которые должны быть расположены на основной странице приложения.

Рисунок 1 — Элементы основной страницы

На странице необходимо разместить следующие элементы:
- заголовок,
- игровое поле,
- подсказку текущего действия,
- кнопки действий *Слово*, *Записать*, *Пропуск*, *Отмена*, *Новая*,
- алфавит для выбора букв мышкой,
- введенные игроком или игроками слова,
- другие элементы при необходимости.

Приведенный примерный внешний вид не является обязательным. Вы сами решаете, какое оформление будет в вашем приложении.

При этом желательно обеспечить следующее:

1) содержание желательно разместить по центру страницы.

2) размер страницы не должен быть большим, чтобы не появлялись линейки прокрутки.

3) при изменении размера окна браузера расположение элементов не должно изменяться.

1.4. Начало работы

Начинать нужно с создания файлов страницы и таблицы стилей.

Заметим, что страница HTML состоит из заголовочной части, располагаемой в теге head, и отображаемого содержания, располагаемого в теге body:

```
<html>
<head>
 <!-- заголовочная часть -->
</head>
<body>
 <!-- отображаемая часть -->
</body>
</html>
```

В заголовочной части записывается различная управляющая информация, например, указание кодировки и таблицы внешних стилей. Все, что должно быть видно на странице пользователю, записывается в отображаемой части.

Файл страницы должен иметь имя index.html, в файл таблицы стилей будет иметь имя index.css. Совпадение имен случайное, не обязательное.

Для работы используем приложение FAR Manager.

Сначала переходим в корневой каталог, введя сочетание Ctrl+\.

Затем создаем папку ksg. Нажимаем F7, вводим название папки ksg и нажимаем Enter. После создания папки заходим в нее, дополнительно нажав клавишу Enter.

Чтобы создать текстовый файл, вводим Shift+F4, затем печатаем название файла index.html и нажимаем Enter.

Записываем в файл следующее содержание:

```
01 <!DOCTYPE html>
02 <html>
03 <head>
04  <title>Королевский квадрат</title>
05  <meta charset='utf-8'/>
06  <link rel='stylesheet' type='text/css' href='index.css'/>
07 </head>
08 <body>
09 </body>
10 </html>
```

В строке 1 задается тип документа HTML5.

В строке 4 задается заголовок, отображаемый в браузере.

В строке 5 задается кодировка страницы.

В строке 6 указывается внешняя таблица стилей.

Между строчками 8 и 9 далее будем размещать содержание.

Сохраняем файл при помощи F2, и закрываем редактор Escape.

Снова открываем файл, вводим Shift+F8 и выбираем кодировку utf-8.

Закрываем файл и проверяем, что в начале файла нет символов BOM.

Для этого открываем файл клавишей F3 и смотрим, какие символы находятся в начале файла, выбрав режим дампа при помощи клавиши F4.

Дальнейшее редактирование файла выполняется клавишей F4.

Теперь аналогичным образом создаем файл таблицы стилей index.css.

На первом этапе записываем в стили следующее:

```
body,td {
  font-family: serif;
  font-size: 13pt;
  padding: 0;
  margin: 0;
}
```

Сохраняем F2 и закрываем файл Escape. Кодировка таблицы стилей не имеет значения до тех пор, пока в ней нет символов, не входящих в первые 128 символов кодировки ASCII (пока в ней нет символов кириллицы).

Суть записанных конструкций следующая.

Элементы страницы body и td по умолчанию будут отображать текст шрифтом с засечками размером 13 пунктов, при этом у них не будет ни отступов (padding), ни полей границ (margin). На рисунке 2 показан смысл отступов и полей.

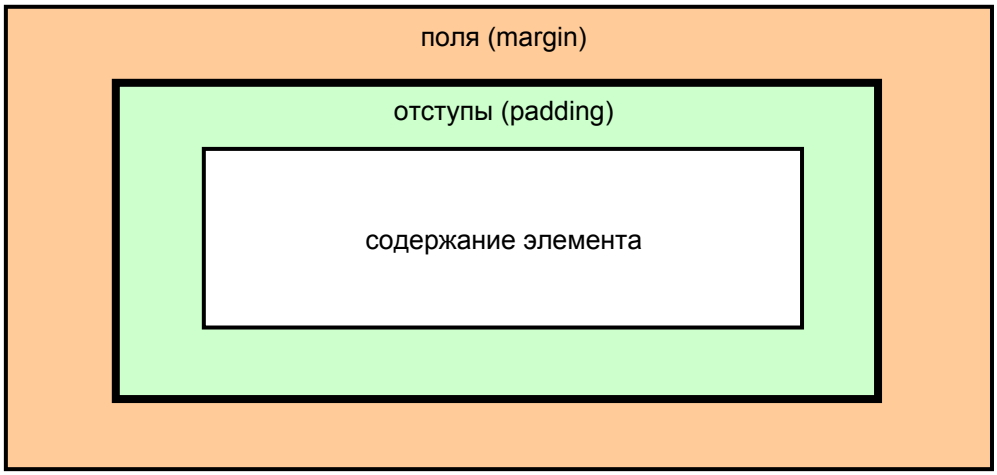

Рисунок 2 — Поля и отступы элементов HTML

Если у элемента есть рамка, то она располагается между полями и отступами. На рисунке рамка показана утолщенной линией. Чаще используются отступы. Они задают пространство между рамкой и содержанием.

Поля задают положение данного элемента по отношению к соседям.

Заметим, что поля и отступы могут быть различными с разных сторон.

Обратим внимание на то, что единица измерения px или pt должна записываться без пробела после числового значения. В приложении FAR Manager, если вы не сделаете этого, значение атрибута поменяет цвет, предупреждая о неправильной записи.

Для записи других отдельных отступов и полей используются аналогичные конструкции (показаны примеры записей):

```
padding-top: 2px;
padding-right: 4px;
padding-bottom: 8px;
padding-left: 16px;
margin-top: 32px;
margin-right: 64px;
margin-bottom: 128px;
margin-left: 256px;
```

Обычно для задания полей и отступов, если они разные с разных сторон, используется следующая конструкция (показан пример записи):

```
padding: 2px 4px 8px 16px;
```

Нужно запомнить, что порядок задания отступов и полей следующий: *сверху*, *справа*, *снизу* и *слева*, то есть по часовой стрелке от верха (*top*).

Таким образом, приведенная выше конструкция задает отступ сверху 2 пикселя, а слева — 16 пикселей.

Если все отступы одинаковые, можно указывать только одно значение, например:

```
padding: 2px;
padding: 0;
```

Вернемся к нашей странице.

Чтобы посмотреть результат, в странице нужно создать содержание.

Открываем файл страницы при помощи F4 и добавляем текст:

```
<!DOCTYPE html>
<html>
<head>
 <title>Королевский квадрат</title>
 <meta charset='utf-8'/>
 <link rel='stylesheet' type='text/css' href='index.css'/>
</head>
<body>
 Королевский квадрат
</body>
</html>
```

Теперь открываем страницу при помощи какого-либо браузера. В приложении FAR Manager для этого достаточно нажать Enter, когда файл страницы выбран. При этом откроется браузер по умолчанию.

Видим пустую страницу, на которой только одна строка, расположенная точно вверху слева. При этом браузер, скорее всего, выбрал шрифт Times New Roman.

Есть одно обстоятельство, затрудняющее разработку страниц HTML.

Дело в том, что разные браузеры могут по-разному их отображать.

Это актуально для устаревших браузеров, и в большой степени для браузера Microsoft Internet Explorer, в просторечии IE. Поэтому во время

разработки страницу желательно открывать во всех доступных браузерах и проверять, как они относятся к тому, что вы написали.

В последние годы большинство разработчиков популярных браузеров вовремя отслеживают современные тенденции, и стараются сделать их примерно одинаковыми. Тем не менее, различия существуют, и их нужно как-то учитывать.

Один из вариантов, — это игнорировать устаревшие и малоизвестные браузеры. Тогда пользователи этих браузеров могут видеть вместо того, что вы задумали, совершенно невообразимые вещи.

Другой вариант, — пытаться найти обходные решения для отдельных типов браузеров. В сети Интернет можно найти немало различных предложений по обеспечению совместимости.

Третий вариант, — никогда не использовать решения, которые ведут к разной интерпретации в разных браузерах. Я предпочитаю этот вариант, хотя при этом не всегда удается достичь желаемого результата. В результате иногда приходится комбинировать этот способ со вторым.

При этом нужно также учитывать, что различия между браузерами постепенно сглаживаются. Несколько лет назад ситуация с браузерами была очень актуальна, если не сказать катастрофична. Например, браузер IE6 (Internet Explorer 6.0) не может отображать скругленные рамки.

Если думать только о современных версиях браузеров, то, видимо, можно полагаться на их соответствие новым стандартам, как по языкам HTML5 и CSS3, так и по языку JavaScript.

Я использую браузер Yandex, которых по своим характеристикам сопоставим с браузером Google Chrome. Другие похожие браузеры, — это Opera и Mozilla Firefox, причем последний значительно отличается от других в части обработки событий. В сети Интернет можно найти статистику использования браузеров пользователями. Она покажет, какой процент пользователей не будет доволен вашими разработками.

В ходе практических работ мы не можем изучить все тонкости применения HTML и CSS. В помощь нам понадобится сеть Интернет.

Актуальную информация по тегам и атрибутам (параметрам) тегов HTML может предоставить, например, страница http://htmlbook.ru/html/. Здесь же можно найти информацию и по стилям CSS, при этом вы просто переходите на страницу http://htmlbook.ru/css/.

Обращайте внимание на поддержку тех или иных тегов и атрибутов разными популярными браузерами, она указывается, когда вы просматриваете описание элемента. На указанных страницах для каждого элемента можно найти пример программирования, и увидеть, как этот пример отображается, открывая сайт в различных браузерах.

1.5. Разметка страницы

Перед тем, как писать код HTML, нужно хорошо представлять, где и что должно быть размещено на странице, и как это можно сделать. Начинать нужно с разметки страницы, то есть определения областей, в которых будут отображаться те или иные элементы. В соответствии с рисунком 1 общее представление об областях представлено на рисунке 3.

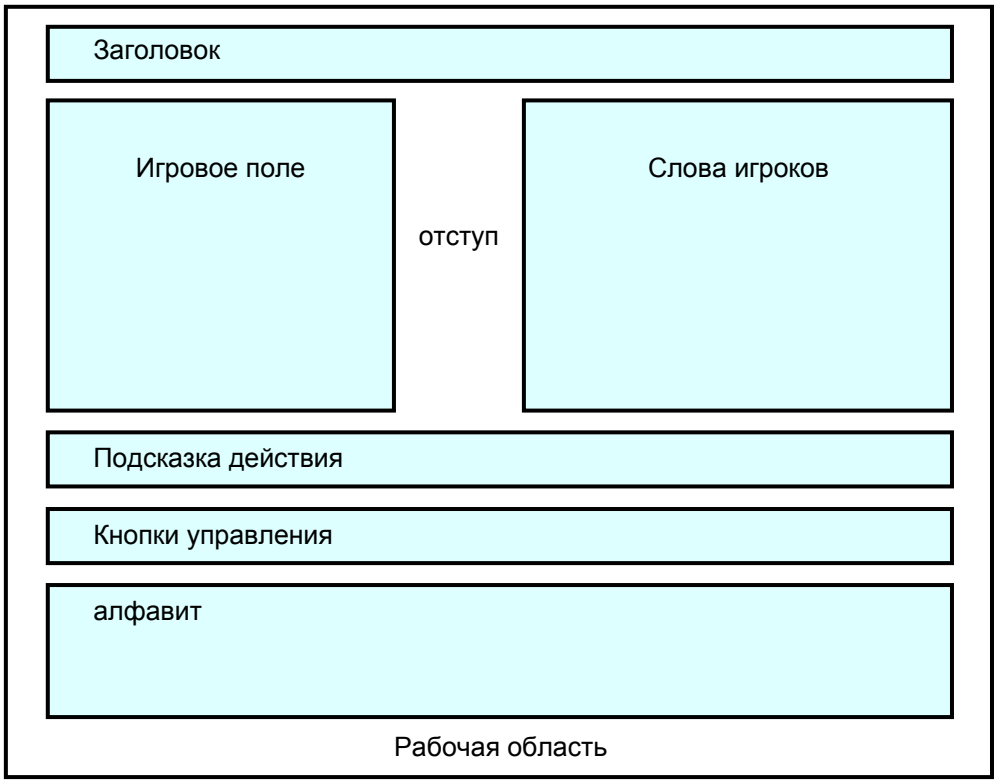

Рисунок 3 — Области размещения содержания

Рабочая область нужна для того, чтобы определить общее положение содержания относительно окна браузера: слева, справа или по центру.

Другие области находятся внутри рабочей области.

Задать области можно при помощи элементов-контейнеров. Наиболее популярные контейнеры, — это блок div и таблица table. У каждого из этих элементов есть свои достоинства и недостатки.

При отсутствии опыта нужно экспериментировать. Пробуем для начала задать рабочую область, которую желательно разместить по центру окна. Определяем блок div в коде страницы:

```
<body>
<div>
 Королевский квадрат
</div>
</body>
```

Обновляем страницу в браузере, видим, что ничего не изменилось.

На этапе разработки желательно видеть разметку, поэтому можно, например, закрасить блок каким-либо цветом. Для этого нужно определить стиль оформления блока. Открываем файл стилей и определяем класс стиля для внешнего блока div:

```
.main {
 background-color: #eef;
}
```

Атрибут background-color задает цвет фона. Применяем стиль к блоку. Изменяем код страницы index.html:

```
<div class='main'>
 Королевский квадрат
</div>
```

Обновляем страницу и видим, что блок по ширине занимает все окно, а по высоте занимает столько места, сколько требуется для размещения содержания, то есть текста «Королевский квадрат». Наверное, это не то, что ожидалось. Размеры блока можно задать в классе оформления. Снова открываем файл стилей index.css и пробуем задать размеры:

```
.main {
 background-color: #eef;
 width: 600px;
 height: 400px;
}
```

Сохраняем файл стилей и обновляем страницу. Блок получил заданный размер, но располагается в левой части страницы. Чтобы разместить блок по центру, нужно добавить в стиль атрибут margin со значением auto:

```
.main {
 background-color: #eef;
 width: 600px;
 height: 400px;
 margin: auto;
}
```

Обновляем страницу и убеждаемся, что блок размещен по центру.

1.5.1. Заголовок и подсказка

Разместим заголовок в блоке div, формат текста блока определим при помощи нового стиля. Открываем файл стилей, описываем стиль head:

```
.head {
 background-color: #fee;
 padding: 8px 0px 8px 0px;
 font-size: 24pt;
 color: #f40;
}
```

Открываем файл страницы и перемещаем заголовок «Королевский квадрат» в блок div:

```
<body>
<div class='main'>
 <div class='head'>Королевский квадрат</div>
</div>
</body>
```

Обновляем страницы в браузерах.

С подсказкой поступим точно так же, только определим другой стиль:

```
.hint {
 background-color: #efe;
 padding: 8px 0px 0px 0px;
 font-size: 13pt;
 text-align: left;
 color: #00f;
}
```

Открываем файл страницы и описываем подсказку «Введите начальное слово» в другом блоке div:

```
<body>
<div class='main'>
 <div class='head'>Королевский квадрат</div>
 <div class='hint'>Введите начальное слово</div>
</div>
</body>
```

Обновляем страницы в браузерах.

1.5.2. Игровое пространство

Игровое пространство, как показано на рисунке 3, состоит из игрового поля, промежутка, и таблицы введенных слов. Разделить ширину рабочей области на три части проще всего при помощи таблицы:

```
<body>
<div class='main'>
 <div class='head'>Королевский квадрат</div>
 <table width='100%' cellspacing='0' cellpadding='0' border='1'>
  <tr><td>1
  </td><td>2
  </td><td>3
  </td></tr>
 </table>
 <div class='hint'>Введите начальное слово</div>
</div>
</body>
```

Тег table описывает таблицу.

Тег tr описывает строку (*table row*).

Тег td описывает ячейку (*table data*).

Закрывать эти теги, в принципе, не обязательно.

Обновим страницы в браузерах. Видим, что таблица появилась и она содержит 3 ячейки с цифрами 1, 2 и 3. Рамки таблицы включены сейчас для того, чтобы видеть области ячеек.

В первой из ячеек нужно разместить игровое поле. Это таблица размером 5×5 ячеек, ячейки будут содержать буквы, желательно покрупнее. Сначала определим стиль для ячейки, открываем файл стилей и описываем следующий стиль:

```
.cell {
 width: 40px;
 height: 40px;
 font-size: 18pt;
 text-align: center;
 cursor: default;
}
```

Теперь нужно описать таблицу из 5 строк и 5 ячеек в строке.

Открываем файл страницы.

Сначала удалим рамки внешней таблицы и цифру 1:

```
<table width='100%' cellspacing='0' cellpadding='0' border='0'>
 <tr><td>
 </td><td>2
 </td><td>3
 </td></tr>
</table>
```

Задаем ширину ячейки для таблицы игрового поля, равной 212 пикселей из расчета: 5×40 = 200 + 2×6 = 212. Вместо цифры 1 вписываем следующую таблицу, части которой получаются копированием:

```
<table width='100%' cellspacing='0' cellpadding='0' border='0'>
 <tr><td width='212px'>
  <table cellspacing='0' cellpadding='0' border='1'>
  <tr>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
  </tr><tr>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
  </tr><tr>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
  </tr><tr>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
```

```
  </tr><tr>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
   <td class='cell'>A</td>
  </tr>
  </table>
  </td><td>2
  </td><td>3
 </td></tr>
</table>
```

Зададим ширину промежутка равной, например, 56 пикселям.

Сделаем это непосредственно в описании второй ячейки, она теперь находится в самом конце страницы:

```
  </td><td width='56px'>2
 </td><td>3
 </td></tr>
</table>
```

После этого можно удалить цифру 2, записав вместо нее неразрывный пробел " ", а цифра 3 пусть висит на странице, как напоминание, что здесь что-то нужно сделать:

```
  </td><td width='56px'> 
  </td><td>3
 </td></tr>
</table>
```

Обновим страницы в браузерах. Примерный вид страницы к настоящему моменту показан на рисунке 4.

Рисунок 4

Часть 3 таблицы пока оставим до лучших времен.

1.5.3. Блок кнопок

Кнопки лучше разместить внутри таблицы из пяти ячеек. Деля 100% ширины таблицы на 5, получим 20% ширины на кнопку. Выравнивая одни ячейки по левому краю, а другие по правому краю, получим необходимое размещение. Для ячеек кнопок создаем стиль butn, который задает отступы сверху и снизу, равные 8 пикселям. Открываем файл стилей и описываем стиль butn для ячеек кнопок и стиль button самих кнопок:

```
.butn {
 padding: 8px 0px 8px 0px;
}
button {
 width: 100px;
 height: 28px;
 background-color: #fff;
 border: 1px solid #000;
 border-radius: 4px;
 color: #000;
}
```

Формируем таблицу для размещения кнопок. Открываем файл страницы и вписываем таблицу после блока подсказки:

```
  </td><td>
  </td></tr>
 </table>
 <div class='hint'>Введите начальное слово</div>
 <table width='100%' cellspacing='0' cellpadding='0' border='1'><tr>
  <td class='butn' width='20%'>
  </td><td class='butn' width='20%'>
  </td><td class='butn' width='20%'>
  </td><td class='butn' width='20%'>
  </td><td class='butn' width='20%'>
 </tr></table>
</div>
</body>
</html>
```

Теперь нужно вписать в каждую ячейку таблицы кнопку, например:

```
  <td class='butn' width='20%'><button>Слово</button>
  </td><td class='butn' width='20%'><button>Записать</button>
```

Кнопки выглядят примерно так, как показано на рисунке 5.

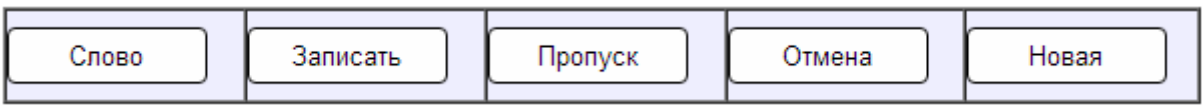

Рисунок 5

Снова открываем файл страницы. Удаляем рамки этой таблицы, устанавливая параметр border в 0. Затем в три последние ячейки td записываем выравнивание по правому краю при помощи параметра align:

```
<table width='100%' cellspacing='0' cellpadding='0' border='0'><tr>
 <td class='butn' width='20%'><button>Слово</button>
 </td><td class='butn' width='20%'><button>Записать</button>
 </td><td class='butn' width='20%' align='right'><button>Пропуск</b
 </td><td class='butn' width='20%' align='right'><button>Отмена</bu
 </td><td class='butn' width='20%' align='right'><button>Новая</but
```

Результат показан на рисунке 6.

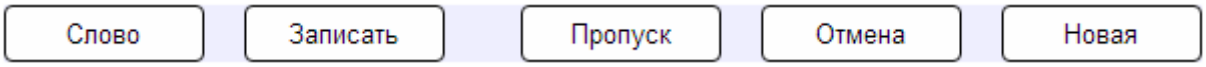

Рисунок 6

1.5.4. Поле алфавита

Наконец, остается нарисовать буквы алфавита. Эта задача ничем принципиально не отличается от рисования игрового поля, поэтому здесь не описывается. Замечу только, что для ячеек таблицы нужно задать стиль alfa, аналогичный стилю cell, только размер ячеек нужно сделать равным 36 пикселям, тогда ширина этой таблицы составит 610 пикселей (16 ячеек в строке шириной 36 пикселей плюс 17 рамок по 2 пикселя).

1.6. Завершение

Нужно внести следующие изменения в файл стилей:

- в стиле main установить ширину 610 пикселей, а высоту удалить, она сама станет равной высоте всего содержимого;
- убрать цвет фона из всех стилей (атрибут background-color), так как он был нужен для оценивания разметки;
- заменить буквы А на игровом полем неразрывным пробелом , так как буквы были нужны также только для оценивания стиля.

2. Работа JS-02. Введение в JavaScript (2 часа)

Цели:
- введение JavaScript в страницу HTML.
Задачи:
- получение ссылок на объекты страницы;
- управление содержанием объектов;
- выделение ячеек таблиц.

2.1. Размещение сценария JavaScript

Сценарий JavaScript может быть размещен в любом месте страницы, выберем для него конец страницы, за закрывающим тегом html. Открываем файл страницы, записываем тег сценария и функцию, которая включится в момент завершения загрузки документа:

```
</body>
</html>
<script>
window.onload = function() {
  alert('start')
}
</script>
```

Здесь используется событие окончания загрузки load объекта window. Функция alert используется нами для отладки. Обновим страницу в браузере и убедимся, что сообщение «start» выводится в виде диалогового окна. Это значит, что функция сработала и можно двигаться дальше.

2.2. Получение ссылки на объект

Ссылка на объект нужна для того, чтобы или узнавать его параметры, или изменять их. Чтобы получить ссылку, нужно в теге объекта записать параметр id, а затем использовать метод getElementById. Нам понадобится менять значение подсказки пользователю, с этого объекта и начнем.

Находим в странице HTML тег div, в котором размещается текст подсказки из записываем в тег параметр id, равный hdiv:

```
 <div id='hdiv' class='hint'>Введите начальное слово</div>
```

Переходим в window.onload. Перед методом alert вписываем получение ссылки на подсказку, а в методе alert изменяем параметр (вместо строки "start" переменная hdiv):

```
window.onload = function() {
  hdiv = document.getElementById('hdiv')
  alert(hdiv)
}
```

Рекомендуется также включить инструменты разработчика в браузере, это позволит увидеть необработанные исключительные ситуации.

Обновляем страницу в браузере и убеждаемся, что выводится сообщение [obect HTMLDivElement]. Это означает, что ссылка есть. Если выводится null (пусто), ссылки нет, в этом случае, вероятно, где-то указан не тот идентификатор hdiv, либо название функции написано с ошибкой.

2.3. Изменение текста тега

Следующая задача — изменить текст подсказки. Это делается через свойство с названием innerHTML (внутренний код HTML. Есть еще внешний код HTML, включающий в себя то, что вокруг текста, то есть теги).

Сначала пробуем получить текст подсказки в переменную s:

```
  hdiv = document.getElementById('hdiv')
  var s = hdiv.innerHTML
  alert(s)
```

Тестируем этот код и убеждаемся, что текст подсказки выводится в диалог сообщения.

Теперь пробуем изменить текст подсказки:

```
window.onload = function() {
  hdiv = document.getElementById('hdiv')
  hdiv.innerHTML = 'Это подсказка юзеру'
}
```

Убеждаемся, что текст на странице изменяется.

2.4. Исключительные ситуации

Изменим одну букву в коде JS:

```
window.onload = function() {
  hdiv = document.getElementById('hduv')
  hdiv.innerHTML = 'Это подсказка юзеру'
}
```

Здесь буква i слова hdiv заменена буквой u. Тестируем этот код. Ничего не работает, а если включены инструменты разработчика, вы увидите, что появилось необработанное (uncaught) исключение.

Ошибки часто случаются во время разработки.

В отлаженном приложении они не должны возникать, но любое изменение кода чревато новой ошибкой. Есть два способа устранения ошибок, хоть на стадии разработки, хоть на стадии эксплуатации. Первый способ заключается в отлавливании исключительных ситуаций при помощи try:

```
window.onload = function() {
  hdiv = document.getElementById('hduv')
 try {
  hdiv.innerHTML = 'Это подсказка юзеру'
 } catch(e) { alert(e) }
}
```

Тестируем и убеждаемся, что выводится сообщение об ошибке.

В приложении сообщение не должно выводиться пользователю, но на стадии отладки может быть полезно. Другой способ — убедиться в том, что используется свойство действительного объекта:

```
window.onload = function() {
  hdiv = document.getElementById('hduv')
  if (hdiv != null && hdiv != undefined)
  hdiv.innerHTML = 'Это подсказка юзеру'
}
```

Здесь ошибки не возникает, так как условие if не выполняется.

Исправим неправильно написанный идентификатор hduv на hdiv.

Перед window.onload опишем вспомогательную функцию для получения ссылки на объект, чтобы не писать все время одно и то же:

```
function ob(id) { return document.getElementById(id) }
window.onload = function() {
  . . .
}
```

Теперь заменим метод document.getElementById:

```
window.onload = function() {
  hdiv = ob('hdiv')
  if (hdiv != null && hdiv != undefined)
  hdiv.innerHTML = 'Это подсказка юзеру'
}
```

Сохраним и обновим страницу. Убедимся, что подсказка изменяется.

2.5. Обработчики событий

Некоторые из объектов страницы являются активными в том смысле, что они реагируют на определенные действия пользователя. Для этого с этими объектами связывают обработчики событий. Обработчик события *xyz* — это свойство-функция on*xyz*, которая будет вызвана при наступления события. Есть несколько способов связать функцию с событием.

2.5.1. Установка обработчика в теге HTML-элемента

Первый способ заключается в том, чтобы определить обработчик непосредственно в теге объекта. Если мы хотим, чтобы при наведении мыши на подсказку она выделялась цветом фона, то:

- нужно определить функцию, которая выполнит выделение;
- в теге описать атрибут (свойство) onmouseover.

Попробуем сделать это. Опишем функцию выделения (после ob):

```
function select() { hdiv.style.background = '#ac0' }
```

Переходим к блоку div и вписываем обработчик:

```
 <div id='hdiv' class='hint' onmouseover='select()'>. . .</div>
```

Сразу после этого наведение мыши на подсказку изменит ее фон.

Заметим, что действия функции можно записать прямо в теге:

```
<div id='hdiv' class='hint'
      onmouseover='this.style.background = "#ac0"'>. . .</div>
```

Здесь используется указатель this, который указывает на тег.

Недостаток функции select в том, что она выделяет один объект. Опишем функцию, которая уберет выделение любого объекта (после ob):

```
function unsel(o) { o.style.background = '' }
```

Обратим внимание на то, как удаляется фон объекта.

Вернемся к тегу блока и установим еще один обработчик:

```
<div id='hdiv' class='hint'
     onmouseover='this.style.background = "#ac0"'
     onmouseout='unsel(this)'>. . .</div>
```

Убеждаемся, что фон исчезает, когда мышь уходит с объекта.

Этот способ кажется простым и удобным. Однако он в значительной степени загромождает код HTML, препятствуя легкому его прочтению дизайнерами web-страниц. В последнее время такая установка обработчиков считается признаком дурного тона.

2.5.2. Обработчик — поименованная функция

Рекомендуемым способом установки обработчика события является программный. Важно понимать, что независимо от способа, результат установки обработчика будет одинаков, — у объекта появится свойство с названием onназвание-события, например, onclick или onmousedown.

На самом деле между двумя способами установки обработчика есть следующая разница. При установке обработчика в теге требуется поименованная функция. При программной установке используется как поименованная функция, так и безымянная, или анонимная функция. Второе отличие, — при программной установке функции нельзя передать ссылку this как параметр, — функция не связана с конкретным тегом.

Восстановим первоначальный вид тега div подсказки, то есть удалим из тега установку обработчиков событий.

```
<div id='hdiv' class='hint'>Введите начальное слово</div>
```

Исследуем теперь, как установить обработчик события программным способом. Перейдем в метод window.onload. Кстати говоря, сам этот метод есть не что иное, как свойство onload, — то есть обработчик события load. И, по определению, это не метод, а свойство.

Сначала установим обработчик наведения мыши:

```
window.onload = function() {
  hdiv = ob('hdiv')
  hdiv.onmouseover = select
}
```

В данном примере в качестве обработчика устанавливается поименованная функция, которую использовали и раньше. Обратим внимание, что при назначении функции таким способом нельзя указывать скобки после названия функции select. Если это сделать, вместо установки обработчика будет выполнена указанная функция.

Убеждаемся, что обработчик установлен и подсказка выделяется.

Далее пробуем установить второй обработчик:

```
window.onload = function() {
  hdiv = ob('hdiv')
  hdiv.onmouseover = select
  hdiv.onmouseout = unsel
}
```

Поскольку функции unsel параметр не передается, в описании функции он и не нужен. Однако стоит проверить, чему он будет равен при входе в функцию:

```
function unsel(o) {
  alert(o)
  o.style.background = ''
}
```

Сохраняем файл и обновляем страницу. Проводим мышь над подсказкой и убеждаемся, что выводится сообщение [object MouseEvent]. То есть, если в функции, назначаемой таким образом, объявить параметр, то он принимает значение объекта event. Естественно, что последующий код

```
  o.style.background = ''
```

является при этом бессмысленным.

Однако для такой функции справедливо и следующее: внутри функции обработчика события доступна ссылка this, указывающая на объект, в котором событие произошло.

Поэтому пробуем заменить объект *o* объектом this:

```
function unsel(o) {
  alert(o)
  alert(this)
  this.style.background = ''
}
```

Сохраняем файл и обновляем страницу.

Действительно, ссылка this имеет место быть внутри данной функции. Вызов метода alert(this) выводит [object HTMLDivElement], и выделение цветом исчезает.

Однако не все так просто. Пробую проверить все то же самое в контрольном браузере IE6. Оказывается, что вызов alert(*o*) выводит значение undefined, а вызов alert(this) выводит object. Иначе говоря, IE6 не передает функции объект event, но внутри функции есть объект this. Выделение фоном в IE6, соответственно, также удаляется.

2.5.3. Обработчик — анонимная функция

Исследуем теперь установку обработчика при помощи анонимной функции. Нас интересуют параметры, которые можно передать такой функции, в частности, анонимный параметр this.

Пробуем в функции window.onload установить анонимный обработчик, который выделит подсказку при наведении мыши:

```
window.onload = function() {
  hdiv = ob('hdiv')
  hdiv.onmouseover = function(e) {
    alert(e)
    this.style.background = '#ac0'
  }
  hdiv.onmouseout = function(e) {
    this.style.background = ''
  }
}
```

Сохраняем файл и обновляем страницу. Выделение и удаление выделения происходит, ссылка this передается. Функция alert(e) показывает, что объектом является событие. Все так же, как и при использовании поименованной функции. В одних браузерах event глобальный объект, в других он передается неявно функции обработчика, и тогда для получения события нам нужна следующая функция:

```
function getEvent(e) { return e ? e : event }
function ob(id) { return document.getElementById(id) }
```

Работает она следующим образом. Если параметр e определен, значит параметр является event, иначе должен быть доступен глобальный объект event.

Остается также вопрос, как в функции обработчика получить объект, вызвавший событие. Ответ дает изучение свойств объекта event.

У объекта event есть два свойства, указывающие на источник. Первое свойство называется target, а второе — srcElement. Одни браузеры имеют первое свойство, другие второе. По аналогии с функцией getEvent можно описать функцию getSource:

```
function getSource(e) { return e.target ? e.target : e.srcElement }
function getEvent(e) { return e ? e : event }
```

Таким образом, у нас есть функции, позволяющие получить как событие, так и источник события, и мы переходим к выделению ячеек таблиц.

2.6. Выделение букв алфавита

Нам нужно, чтобы при щелчке мышью на букву алфавита она получала подсветку. Во-первых, это событие click. Во-вторых, выделение одной буквы должно снимать выделение уже выделенной буквы. В-третьих, выбранная буква должна как-то запоминаться.

Изменим функцию выделения:

```
function ob(id) { return document.getElementById(id) }
function select(o, b) { o.style.background = b }
```

Она объекту *o* задает некоторый фон *b*.

Опишем функцию для инициализации таблицы алфавита:

```
function select(o, b) { o.style.background = b }
function initA() {
}
```

Вызовем эту функцию в window.onload:

```
window.onload = function() {
  initA()
}
```

Нам потребуется ссылка на объект таблицы алфавита и на первую ее ячейку, зададим для этих объектов идентификаторы:

```
 <table id='atbl' cellspacing='0' cellpadding='0' border='1'>
 <tr>
  <td id='a0' class='alfa'>A</td>
```

Пусть есть свойство window.cua (current alfa), которое хранит объект текущей выбранной ячейки алфавита. Тогда можно описать обработчик события click ячеек алфавита следующим образом:

```
function initA() {
  ob('atbl').onclick = function(e) {
    select(cua, '')
    cua = getSource(getEvent(e))
    select(cua, '#fcf')
  }
  cua = ob('a0')
}
```

Обработчик использует механизм всплывания событий. Если событие произошло в объекте и не обработано в нем, оно передается вверх по иерархии объектов до тех пор, пока не будет обработано, или пока не умрет. Это дает возможность обрабатывать события ячеек таблицы, отлавливая их в обработчике тега table. Свойство cua изначально содержит объект первой ячейки таблицы, то есть ячейки буквы A.

Сохраняем файл и обновляем страницу. Убеждаемся, что ячейки выделяются и выделение перемещается от ячейки к ячейке.

2.7. Выделение ячеек игрового поля

Механизм выделения ячеек игрового поля будет таким же. Сначала зададим идентификаторы для таблицы и первой ее ячейки:

```
  <tr><td width='212px'>
   <table id='ctbl' cellspacing='0' cellpadding='0' border='1'>
   <tr>
    <td id='c0' class='cell'> </td>
```

Описываем функцию инициализации игрового поля:

```
function initA() {
  ob('atbl').onclick = function(e) {
    select(cua, '')
    cua = getSource(getEvent(e))
    select(cua, '#fcf')
  }
  cua = ob('a0')
}
function initC() {
  ob('ctbl').onclick = function(e) {
    select(cuc, '')
    cuc = getSource(getEvent(e))
    select(cuc, '#fc8')
  }
  cuc = ob('c0')
}
```

Здесь вместо объекта atbl используется объект ctbl, вместо метки a0 метка c0, вместо свойства cua свойство cuc (current cell).

Вызываем initC в window.onload:

```
window.onload = function() {
  initA()
  initC()
}
```

Сохраняем файл и обновляем страницу.

Убеждаемся, что ячейки игрового поля также выделяются.

Дополнительно рекомендуется почитать страницу:
https://learn.javascript.ru/introduction-browser-events.

3. Работа JS-03. Ввод начального слова (2 часа)

Цели:
- управление ячейками.
Задачи:
- состояния сценария;
- управление ячейками;
- инициализация сеанса игры;
- ввод начального слова.

Наша первоочередная задача в смысле игрового момента — ввести начальное слово. Прежде нужно ввести понятие состояния игры.

3.1. Состояния приложения

Очевидными состояниями приложения являются:
- ввод начального слова INIT,
- выбор ячейки для новой буквы CELL,
- выбор ячеек нового слова WORD,
- завершение игры OVER.

На самом деле требуется еще состояние, когда сценарий выполняет длительный процесс и действия пользователя в этот момент нежелательны, назовем состояние IDLE. В соответствии с этим определим константы состояний и переменную для текущего состояния:

```
<script>
var IDLE=0, INIT=1, CELL=2, WORD=3, OVER=4, state=INIT
```

3.2. Подсказки пользователю

Определим функцию, которая изменит текст подсказки:

```
function hint(m) { ob('hdiv').innerHTML = m }
```

Определим функцию, которая меняет состояние сценария:

```
function setState(s) {
  state = s;
  switch (s) {
  case INIT:
    hint('Введите буквы начального слова')
    break
  case CELL:
    hint('Выберите новую ячейку, введите букву и нажмите «Слово»')
    break
  case WORD:
    hint('Укажите буквы нового слова и нажмите «Записать»')
    break
  case OVER:
    hint('Игра завершена')
  }
}
```

3.3. Управление ячейками игрового поля

В начальном состоянии щелчок в ячейку алфавита должен записывать букву ячейки в последовательные ячейки игрового поля, имеющие относительные индексы 10, 11, 12, 13 и 14. Возникает вопрос, как получить доступ к ячейке игрового поля по ее индексу.

Простое решение заключается в том, чтобы создать массив объектов, в котором отдельный элемент соответствует объекту ячейки поля, но это решение плохое — хранить множество объектов является роскошью и недопустимым использованием ресурсов. Нужен какой-то другой механизм.

Для получения ссылок на объекты есть еще одна функция, она возвращает объекты по имени тега: getElementsByTagName. Нас интересуют ячейки таблиц, теги td. Опишем функцию, которая возвращает объект тега td по абсолютному порядковому номеру:

```
function ob(id) { return document.getElementById(id) }
function td(i) { return document.getElementsByTagName('TD')[i] }
```

Обратим внимание, в имени этой функции слово Elements во множественном числе, не пропустите букву s в конце слова. Это означает, что функция возвращает коллекцию всех объектов данных тегов, но функция является индексируемой. Если нам известен порядковый номер тега в документе, то при помощи функции td мы можем получить объект:

```
window.onload = function() {
  initA()
  initC()
  td(1).innerHTML = 'A'
}
```

Сохраним и обновим страницу.

Убедимся, что в первой ячейке поля появилась буква.

Теперь нам нужно быть уверенными, что первая ячейка поля имеет определенный индекс, например 1. Эта уверенность должна основываться на вычислении номера этой ячейки в начале сеанса. Тогда мы сможем изменять код HTML в смысле изменения таблиц, а код JS вычислит новый индекс ячейки c0. Хранить индекс первой ячейки будем как свойство zero объекта window. Чтобы вычислить его, нужно взять коллекцию всех объектов td и просматривать ее до тех пор, пока не встретится элемент с идентификатором c0.

Определим для этого функцию getIndex:

```
function getIndex(id) {
  var a = document.getElementsByTagName('TD')
  for (var i = 0, n = a.length; i < n; i++) {
    if (a[i].id == id) return i
  }
  alert(id + ' not found') // что-то не так в коде HTML
  return 0
}
```

В случае, если мы что-то испортим, функция нам просигнализирует.

Теперь добавим вызов getIndex в функцию initC:

```
function initC() {
  ob('ctbl').onclick = function(e) {
    select(cuc, '')
    cuc = getSource(getEvent(e))
    select(cuc, '#fc8')
  }
  cuc = ob('c0')
  zero = getIndex('c0')
}
```

Для получения доступа к ячейке поля определим функцию cell:

```
function td(i) { return document.getElementsByTagName('TD')[i] }
function cell(i) { return td(i + zero) }
function select(o, b) { o.style.background = b }
```

При помощи этой функции мы можем выделить первую ячейку слова:

```
window.onload = function() {
  initA()
  initC()
  select(cell(10), '#fc8')
}
```

Сохраним и обновим страницу.

Убедимся, что выделена ячейка для первой буквы начального слова.

Однако у нас появилась проблема. Мы теперь вынуждены каждый раз указывать цвет выделения, а это неправильно. Можно, например, описать константы для всех используемых цветов. Есть и другие варианты.

3.4. Замыкание функций JS

Есть интересная особенность JavaScript, — функции могут возвращать функции. При этом возникает вложенность функций и замыкание вложенной функции, которое нужно понять, и это не совсем просто.

Рассмотрим следующий пример вложенности функций:

```
function sum(a) {
  return function(b) {
    return (a + b)
  }
}
```

Запишем эту функцию в сценарии, в функции window.onload запишем следующий ее вызов:

```
window.onload = function() {
  var a = sum(2)(3)
  alert(a)
  return
  . . .
}
```

Довольно необычно, но вместе с тем легко объяснимо.

На самом деле здесь вызывается функция sum и в нее подставляется параметр *a*, например 2. Функция sum возвращает вложенную функцию, поэтому запись sum(2) всего лишь способ вызова вложенной анонимной функции, и далее по синтаксису должны следовать скобки, в которые записывается параметр *b*, например 3.

Сохраним и обновим страницу.

Убедимся, что результат вычисляется корректно.

Интереснее то, что можно сделать с функцией sum.

Изменим код window.onload на следующий:

```
00 window.onload = function() {
01   var plus3 = sum(3)
02   var plus5 = sum(5)
03   var a = plus3(2)
04   var b = plus5(2)
05   alert(a)
06   alert(b)
07   alert(plus3)
08   alert(plus5)
09   alert(plus3 == plus5)
10   var p3 = plus3.toString()
11   var p5 = plus5.toString()
12   alert(p3 == p5)
13   return
14   . . .
15 }
```

Сохраним и обновим страницу.

Убедимся, что выводятся значения *a* и *b*, равные 5 и 7.

Этот пример кода показывает, когда и как формируются замыкания.

Нужно понимать, что plus3 и plus5 — это свойства-функции, причем литерально совершенно одинаковые, о чем свидетельствует строчка 12, выполнение которой возвращает истину.

Однако plus3 и plus5 не равны в строке 9. Это происходит потому что для функций plus3 и plus5 формируются разные замыкания.

Чтобы обеспечить видимость во вложенной функции, которая сработает неизвестно когда, для этой функции формируется замыкание, состоящее из ссылок на всё, что видит внешняя функция в момент выполнения. Когда будет вызвана внутренняя функция, значения будут извлечены из замыкания (которое фиксируется и хранится все время работы сценария).

В замыкании функции plus3 функция sum была завершена в строке 1 со значением переменной *a*, равным 3, а в замыкании функции plus5 функция sum завершилась в строке 2 со значением переменной *a*, равным 5. Поэтому одна и та же литеральная функция возвращает разный результат, — для разных вызовов sum сложились разные замыкания.

Эта особенность JS позволяет писать простой и ясный код.

Удалим функцию sum и весь тестирующий код из window.onload.

3.5. Функции выделения и очистки

Новая версия функции выделения select:

```
function select(b, c) {
  return function(o) {
    o.style.background = b
    eval(c + '= o')
  }
}
```

В замыкании select запоминаются два литерала: цвет фона *b* и имя ячейки *c*. Функция eval складывает имя ячейки с новым ее значением и вычисляет полученный *текст* кода, как будто мы его написали. Это дает возможность не только выделить ячейку, но запомнить ее как текущую.

Для очистки выделения используем функцию unsel:

```
function cell(i) { return td(i + zero) }
function unsel(o) { o.style.background = '' }
function select(b, c) {
```

Теперь нужно создать замыкания для поля и алфавита.

Функция initA, замыкание selA в последней строке:

```
function initA() {
  ob('atbl').onclick = function(e) {
    unsel(cua)
    selA(getSource(getEvent(e)))
  }
  cua = ob('a0')
  selA = select('#fcf', 'cua')
}
```

В функции selA мы как бы замкнули цвет и имя ячейки cua.

Функция initc, замыкание selC в последней строке:

```
function initC() {
  ob('ctbl').onclick = function(e) {
    unsel(cuc)
    selC(getSource(getEvent(e)))
  }
  cuc = ob('c0')
  zero = getIndex('c0')
  selC = select('#fc8', 'cuc')
}
```

Выделяем ячейку 10:

```
window.onload = function() {
  initA()
  initC()
  selC(cell(10))
}
```

Сохраним и обновим страницу.

Можно убедиться, что выделение ячеек по-прежнему работает.

3.6. Ввод начального слова

Переходим к алгоритму ввода начального слова. Запишем в каждую ячейку поля ее индекс от нуля. Тогда если при вводе начального слова выбрана ячейка поля с индексом *i*, то следующая ячейка *i* + 1. Кроме того, каждая ячейка поля должна не только отображать букву, но и помнить, какая буква записана в нее. Исходя из этого нужно, чтобы у ячейки поля были свойства index и letter. Опишем функцию, которая установит свойства ячеек, и определит свойства-функции:

```
function cellProps() {
  for (var i = 0; i < 25; i++) {
    cell(i).index = i
    cell(i).letter = ''
    cell(i).setLetter = function() {
      this.innerHTML = cua.innerHTML
    }
    cell(i).fixLetter = function() {
      this.letter = this.innerHTML
    }
  }
}
```

Эта функция вызывается в конце функции initC.

```
function initC() {
  . . .
  cuc = ob('c0')
  zero = getIndex('c0')
  selC = select('#fc8', 'cuc')
  cellProps()
}
```

Изменяем код обработчика в функции initA. Описываем в нем последовательность действий для ввода букв начального слова:

```
function initA() {
  ob('atbl').onclick = function(e) {
    if (state != CELL && state != INIT) return
    unsel(cua)
    selA(getSource(getEvent(e)))
    cuc.setLetter()              /* покажем букву */
    if (state != INIT) return    /* дальше только INIT */
    cuc.fixLetter()              /* запомним букву */
    unsel(cuc)                   /* уберем выделение */
    selC(cell(cuc.index + 1))    /* выделим следующую */
    if (cuc.index == 15) {       /* условие завершения INIT */
      setState(CELL)          /* переходим к вводу буквы слова */
      cuc.setLetter()         /* какая-то буква в ячейке 15 */
    }
  }
  cua = ob('a0')
  selA = select('#fcf', 'cua')
}
```

Если состояние не INIT, буква появляется в ячейке поля (показываем букву), но не запоминается. Если состояние INIT, буква запоминается, затем меняется текущая ячейка.

Сохраним и обновим страницу. Убедимся, что начальное слово можно ввести и далее можно изменять буквы на поле.

3.7. Инициализация сеанса

Для формирования условий, при которых начинается сеанс новой игры, нужна дополнительная функция. Ее задачи следующие:

1. Очистить ячейки поля.
2. Убрать выделение ячеек алфавита и выбрать букву А.
3. Выбрать начальную ячейку поля 10 в качестве текущей.
4. Установить состояние сценария INIT.

Для очистки ячеек поля требуются две функции.

Следующая функция очистит одну ячейку:

```
function clear(o) {
  o.style.background = ''
  o.innerHTML = ''
  o.letter = ''
}
```

Следующая функция выполнит действие act над ячейками поля:

```
function cells(act) {
  for (var i = 0; i < 25; i++) {
    act(cell(i))
  }
}
```

Это функция высшего порядка, ее параметр является функцией.

Теперь можно сформировать функцию, начинающую сеанс игры:

```
function onGame() {
  cells(clear)
  unsel(cua)
  selA(ob('a0'))
  selC(cell(10))
  setState(INIT)
}
```

Функция window.onload устанавливает обработчики и начинает игру:

```
window.onload = function() {
  initA()
  initC()
  onGame()
}
```

Балансируя содержанием этих двух функций, нам нужно будет в итоге сформировать правильный стартовый код.

4. Работа JS-04. Основные алгоритмы (2-4 часа)

Цели:
- разработка алгоритмов игровых моментов
Задачи:
- вычисление допустимости ячейки новой буквы;
- ввод букв нового слова;
- вычисление допустимости ячейки нового слова.

4.1. Начальная диспозиция

Будет неудобно отлаживать сценарий, если каждый раз вводить начальное слово. Можно просто записать какое-нибудь слово как начальное и изменить состояние сценария. Перейдем в функцию cellProps и добавим еще одно свойство ячейки:

```
function cellProps() {
  for (var i = 0; i < 25; i++) {
    cell(i).index = i
    cell(i).letter = ''
    cell(i).setLetter = function() {
      this.innerHTML = cua.innerHTML
    }
    cell(i).fixLetter = function() {
      this.letter = cua.innerHTML
    }
    cell(i).set = function(a) {
      this.innerHTML = a
      this.letter = a
    }
  }
}
```

Пусть начальным словом будет ГОРОД. Тогда следующая последовательность действий в window.onload установит это слово:

```
window.onload = function() {
  initA()
  initC()
  onGame()
  cell(10).set('Г')
  cell(11).set('О')
  cell(12).set('Р')
  cell(13).set('О')
  cell(14).set('Д')
  cell(15).innerHTML = 'A'
  unsel(cuc)
  selC(cell(15))
  setState(CELL)
}
```

Сохраним и обновим страницу.

Убедимся, что начальное слово введено и ячейки можно выбирать.

4.2. Ввод новой буквы

После того, как на поле есть начальное слово, один из игроков вводит новую букву с тем, чтобы с ее помощью записать новое слово. Когда игрок щелкает в ячейку поля, она либо допустима для ввода в нее буквы, либо нет. Нужно определить условия, разрешающие или запрещающие ввод в ячейку. Для этой цели описываем функцию validC:

```
function validC(i) {
  switch(state) {
  case INIT:
    break
  case CELL:
    break
  case WORD:
    break
  }
  return false
}
```

По умолчанию функция запрещает ячейку, индекс которой равен *i*.

Рассмотрим состояние INIT. Если игрок ввел букву, то допустимыми индексами являются 10 и все индексы до cuc.index:

```
function validC(i) {
  switch(state) {
  case INIT:
    if (i > 9 && i < cuc.index) return true
    break
  case CELL:
    . . .
}
```

Рассмотрим состояние CELL. Ячейка допустима при выполнении двух условий: 1) в ячейке не записана никакая буква и 2) в любой из соседних с ячейкой *i* ячеек записана какая-то буква.

Определим функцию-предикат, соответствующую условию «в ячейке записана какая-то буква»:

```
function isNotEmpty(x) { return (cell(x).letter != '') }
```

Эту функцию можно записать в validC:

```
function validC(i) {
  switch(state) {
    . . .
  case CELL:
    if (isNotEmpty(i)) return false
    break
  case WORD:
    . . .
}
```

Нужно реализовать еще второе условие, которое должно выполняться, если выполняется первое условие.

Для этого нужно проверять все соседние ячейки. Если поле имеет размер 5×5, то соседними с ячейкой *i* являются ячейки *i*±1, *i*±4, *i*±5, *i*±6, при том условии, что ячейка *i* находится в середине поля.

Если ячейка находится на границе поля, то нужно убедиться также, что индекс соседней ячейки больше или равен нулю и меньше или равен 25, а также что ячейки *i* и *i*±*d* находятся в соседних колонках.

Для вычисления колонки ячейки *i* опишем функцию col:

```
function col(i) { return (i % 5) }
```

Тогда второе условие описывает функция neighborOf:

```
function neighborOf(i, cf, y) {
   var d = [-6,-5,-4,-1,1,4,5,6], a = col(i)
   for (var j = 0; j < 8; j++) {
      var x = i + d[j]
      if (x >= 0 && x < 25 && Math.abs(a - col(x)) < 2) {
         if (cf(x, y)) return true
      }
   }
   return false
}
```

Здесь параметр *i* — проверяемая ячейка, cf — функция-предикат сравнения (compare function), *y* — второй параметр функции cf. Заметим, что в JavaScript в функцию можно передавать какие угодно параметры, при этом их не обязательно иметь в описании функции. Воспользуется предикатом isNotEmpty с одним параметром. Передавая ему второй параметр *y*, если он вообще будет передан функции neighborOf, мы не нарушаем никаких правил языка.

Функция neighborOf — это функция высшего порядка. Теперь можно лаконично и просто записать условия допустимости ячейки в состоянии сценария CELL:

```
function validC(i) {
  switch(state) {
  case INIT:
    if (i > 9 && i < cuc.index) return true
    break
  case CELL:
    if (isNotEmpty(i)) return false
    if (neighborOf(i, isNotEmpty)) return true
    break
  case WORD:
    break
  }
  return false
}
```

Для проверки ячеек в состоянии WORD определим другую функцию:

```
function validW(i) {
   return true
}
```

Пока функция возвращает истину. Вызываем ее в validC:

```
function validC(i) {
  switch(state) {
  case INIT:
    if (i > 9 && i < cuc.index) return true
    break
  case CELL:
    if (isNotEmpty(i)) return false
    if (neighborOf(i, isNotEmpty)) return true
    break
  case WORD:
    return validW(i)
  }
  return false
}
```

Перейдем к коду обработчика событий ячеек поля. Его нужно значительно изменить. Сначала нам нужно получить объект *o*, который вызвал событие, так как в объекте *o* находится индекс проверяемой ячейки. Если индекс допустим, то в зависимости от текущего состояния сценария нужно выполнить те или иные действия.

В состоянии INIT нужно убрать выделение текущей ячейки и выбрать ячейку *o*.

В состоянии CELL нужно очистить текущую ячейку, если она задана, установить букву в ячейке *o* и выбрать ее.

Действия, выполняемые в состоянии WORD, пока не известны:

```
function initC() {
  ob('ctbl').onclick = function(e) {
    var o = getSource(getEvent(e))
    if (!validC(o.index)) return
    if (state == INIT) {
      unsel(cuc)
      selC(o)
    } else if (state == CELL) {
      if (cuc) clear(cuc)
      o.setLetter()
      selC(o)
    } else if (state == WORD) {

    }
  }
  cuc = ob('c0')
  zero = getIndex('c0')
  selC = select('#fc8', 'cuc')
  cellProps()
}
```

Сохраним и обновим страницу.

Убедимся, что выбираются допустимые ячейки и буква переносится при щелчке в другую допустимую ячейку.

4.3. Функции кнопок

Нам требуются обработчики событий кнопок.

Сначала зададим кнопкам идентификаторы. Найдем таблицу с кнопками и в каждый тег button впишем ее идентификатор:

```
<table width='100%' cellspacing='0' cellpadding='0' border='0'><tr>
 <td class='butn' ...><button id='cmdW'>Слово</button>
 </td><td class='butn' ...><button id='cmdS'>Записать</button>
 </td><td class='butn' ...><button id='cmdM'>Пропуск</button>
 </td><td class='butn' ...><button id='cmdB'>Отмена</button>
 </td><td class='butn' ...><button id='cmdN'>Новая</button>
</tr></table>
```

Обработчики будет устанавливать функция initB:

```
function initB() {
  ob('cmdN').onclick = function(e) { onGame() }
  ob('cmdW').onclick = function(e) { onWord() }
  ob('cmdS').onclick = function(e) { onSave() }
  ob('cmdM').onclick = function(e) { onMiss() }
  ob('cmdB').onclick = function(e) { onBack() }
  }
}
```

Здесь указаны новые функции, содержание которых пока не ясно:

```
function onBack() {}
function onMiss() {}
function onSave() {}
function onWord() {}
```

Функцию initB вызываем в window.onload:

```
window.onload = function() {
  initA()
  initC()
  initB()
  onGame()
  . . .
}
```

Сохраним и обновим страницу.

4.4. Цепочка нового слова

Выбрав новую букву, игрок нажимает кнопку «Слово». Сценарий переходит в состояние WORD, выделение ячеек поля убирается. Игрок последовательно выбирает буквы слова, щелкая на них мышью, буквы выделяются. Завершив ввод, игрок нажимает кнопку «Записать».

Будем записывать слово как индексы ячеек, в которые игрок щелкает, в массив с названием chain (цепочка). Нам также нужен механизм, позволяющий узнать, что новая буква введена на поле. Если новая буква не введена, то нельзя вводить новое слово, так как слово должно содержать ее.

Возникает проблема сохранения новой буквы, введенной на поле. Решим ее следующим образом. Когда игрок выбирает буквы нового слова, не будем менять свойство cuc, вместо этого будем менять свойство cuw. Для этого определим еще одно замыкание функции select:

```
function initC() {
  ob('ctbl').onclick = function(e) {
    var o = getSource(getEvent(e))
    if (!validC(o.index)) return
    if (state == INIT) {
      unsel(cuc)
      selC(o)
    } else if (state == CELL) {
      if (cuc) clear(cuc)
      o.setLetter()
      selC(o)
    } else if (state == WORD) {
      chain.push(o.index)
      selW(o)
    }
  }
  cuc = ob('c0')
  zero = getIndex('c0')
  selC = select('#fc8', 'cuc')
  selW = select('#fc8', 'cuw')
  cellProps()
}
```

Функция onWord при этом может иметь следующий вид:

```
function onWord() {
  if (state != CELL && state != WORD) return
  if (cuc == null) return
  if (cuc.innerHTML == '') return
  cells(unsel)
  chain = [cuc.index]
  setState(WORD)
}
```

Функция работает в состояниях CELL и WORD. Второе состояние нужно, когда игрок ошибся при выборе букв слова, и вводит его заново, повторно нажав кнопку «Слово». Цепочка начинается с новой буквы поля. Мы проверяем, что ячейка cuc определена и буква в ней есть. Для контроля цепочки зададим начальный код функции onSave:

```
function onSave() {
  if (state != WORD) return
  if (window.chain == undefined) return
  if (chain.length < 3) return
  alert(chain)
  cells(unsel)
  cuc.fixLetter()
  cuc = null
  setState(CELL)
}
```

Сохраним и обновим страницу. Нажимаем кнопку «Слово», выбираем какие-нибудь буквы, нажимаем кнопку «Сохранить». Убеждаемся, что цепочка формируется. Повторяем действия и убеждаемся, что можно вводить новые буквы на поле и выбирать новые слова.

Однако есть проблема. После нажатия кнопки «Записать» cuc равно null, и выбор буквы алфавита приводит к ошибке. Иначе говоря, после ввода нового слова на поле должна быть выбрана какая-нибудь ячейка.

4.5. Поиск свободной ячейки

Для поиска ячейки для следующей новой буквы определим функцию:

```
function freeCell() {
  setState(CELL)
  for (var i = 0; i < 25; i++) {
    if (validC(i)) return i
  }
  return -1;
}
```

Функция устанавливает состояние CELL, потому что нужно найти ячейку, которая удовлетворяет функции validC. Просматриваем все ячейки поля, и если какая-то из них подходит, возвращаем ее индекс. Если ни одна ячейка не подходит, значит, ячейки закончились, это завершение игры, возвращаем минус один.

Функцию вызываем в onSave:

```
function onSave() {
  if (state != WORD) return
  if (window.chain == undefined) return
  if (chain.length < 3) return
  alert(chain)
  cells(unsel)
  cuc.fixLetter()
  cuc = null
  var free = freeCell()
  if (free < 0) {
    setState(OVER)
  } else {
    selC(cell(free))
    cuc.setLetter()
  }
}
```

Сохраним и обновим страницу.

Убедимся, что после ввода слова ошибок не возникает.

4.6. Функции цепочки

Следующие операции выполняются с цепочкой:

1. Получение слова.
2. Поиск в цепочке некоторой буквы (индекса буквы).

Для этих целей можно использовать одну функцию высшего порядка:

```
function wch(c, pf, a, y) {
   for (var i = 1, k = c.length; i < k; i++) {
      a = pf(a, c[i], y)
   }
   return a
}
```

Название функции wch означает with chain. В функции используется предикат pf (predicate function) с тремя параметрами. Первый параметр предиката *a* — это ожидаемый результат, — либо строка, либо логическое значение. Два следующих параметра — это текущий элемент цепочки c[*i*], и элемент для поиска *y*.

Следующие две функции являются нужными предикатами:

```
function getword(a, x) { return a + cell(x).innerHTML }
function isinchy(a, x, y) { return a || (x == y) }
```

Первая функция собирает слово, вторая — ищет элемент *y*.

Функция getword работает следующим образом. Задается параметр *a*, равный пустой строке. В этом случае операция + поочередно присоединяет к пустой строке буквы, соответствующие индексам цепочки, и результат будет строковым значением.

Функция isinchy работает следующим образом. Задается параметр *a*, равный лжи. В этом случае при обнаружении элемента цепочки, равного искомому *y*, *a* примет истинное значение, которое и будет результатом.

Первое применение функции wch найдется в функции onSave:

```
function onSave() {
  if (state != WORD) return
  if (window.chain == undefined) return
  if (chain.length < 3) return
  if (!wch(chain, isinchy, false, chain[0])) return
  alert(chain)
  cells(unsel)
  cuc.fixLetter()
  cuc = null
  var free = freeCell()
  if (free < 0) {
    setState(OVER)
  } else {
    selC(cell(free))
    cuc.setLetter()
  }
}
```

Функция wch проверяет, что новая буква поля входит в цепочку (а новая буква записана в цепочке под индексом 0).

Сохраним и обновим страницу.

Убедимся, что теперь нельзя записать слово, в которое новая буква поля не входит.

4.7. Допустимые ячейки слова

Перейдем к формированию функции validW. Следующие условия необходимо соблюдать для того, чтобы ячейка была допустимой:

1. Ячейка *i* не должна быть пустой.

Пусть *k* — длина цепочки.

2. Если *k* равно единице, ячейка допустима. В этом случае в цепочке записана только новая буква на поле, и не выбрано ни одной буквы слова, а первой буквой слова может быть любая не пустая ячейка.

3. Ячейка *i* недопустима, если она уже есть в цепочке.

4. Ячейка *i* является соседней с последней в цепочке ячейкой *k* – 1.

5. Ячейка *i* не формирует пересечения цепочки сама с собой.

Если все условия выполняются, ячейка *i* является допустимой.

Некоторые условия просты и их сразу можно записать:

```
function validW(i) {
  if (cell(i).innerHTML == '') return false
  var k = chain.length
  if (k == 1) return true
  if (wch(chain, isinchy, false, i)) return false
  if (!neighborOf(i, equalTo, chain[k - 1])) return false
//  if (cross(i)) return false
  return true
}
```

Здесь используется функция-предикат equalTo:

```
function equalTo(x, y) { return (x == y) }
function isNotEmpty(x) { return (cell(x).letter != '') }
```

Сохраним и обновим страницу.

Остается проверить цепочку только на самопересечение.

4.8. Самопересечение цепочки

На рисунке 7 последняя введенная буква слова chain[*k* – 1] отмечена жирной точкой (ячейка 6), *k* — длина цепочки. Из ячейки 6 можно перейти в ячейки 0, 2, 10 и 12, которые представляют очередную букву слова *i*.

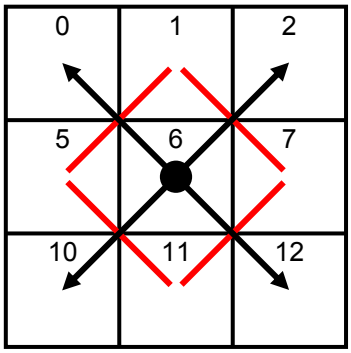

Рисунок 7

Если переход из ячейки 6 к ячейке *i* пересекает красную линию, которая обозначает часть введенной цепочки, возникает самопересечение. Исходя из этого рисунка, можно определить условия самопересечения.

Пусть $d = i - \text{chain}[k - 1]$, $ad = |d|$. Для рисунка $d = i - 6$. Обозначим индексы ячеек, соответствующих концам красных черточек, как c1 и c2.

Случай $i = 0$: $d = -6$, $ad = 6$; $c1 = i + 1$, $c2 = i + 5$, или наоборот.

Случай $i = 2$: $d = -4$, $ad = 4$; $c1 = i - 1$, $c2 = i + 5$, или наоборот.

Случай $i = 10$: $d = 4$, $ad = 4$; $c1 = i + 1$, $c2 = i - 5$, или наоборот.

Случай $i = 12$: $d = 6$, $ad = 6$; $c1 = i - 1$, $c2 = i - 5$, или наоборот.

Пусть $c1 = i + dc1(d)$, $c2 = i + dc2(d)$. Тогда:

$dc1(-6) = 1$, $dc1(4) = 1$, $dc1(-4) = -1$, $dc1(6) = -1$.

$dc2(-6) = 5$, $dc2(-4) = 5$, $dc2(4) = -5$, $dc2(6) = -5$.

Определим функции dc1, dc2, и cross:

```
function dc1(d) {
  switch (d) {
  case -6: case 4: return 1
  case -4: case 6: return -1
  }
}
function dc2(d) {
  switch (d) {
  case -6: case -4: return 5
  case 4: case 6: return -5
  }
}
function cross(i) {
  var k = chain.length - 1
  if (k < 3) return false
  var d = i - chain[k]
  var ad = Math.abs(d)
  if (ad != 4 && ad != 6) return false
  var c1 = i + dc1(d), c2 = i + dc2(d)
  for (var j = 1; j < k; j++) {
    if (chain[j] == c1 && chain[j + 1] == c2) return true
    if (chain[j] == c2 && chain[j + 1] == c1) return true
  }
  return false
}
```

Здесь учитывается, что пересечение не возникает, если в цепочке менее 4 букв, и если *ad* не равно 4 или 6, то есть ход не по диагонали.

В функции validW раскомментируем строку с вызовом функции cross.

Сохраним и обновим страницу.

Убедимся, что пересечение цепочки невозможно ввести. Для этого вводим "А" в ячейку 15, записываем слово РОГА, вводим "З" в ячейку 16, и пробуем записать слово в ячейки: 16—10—15—11, 15—11—10—16, 11—15—16—10 и 10—16—15—11. Эти комбинации недопустимы, как и некоторые другие.

4.9. Вывод нового слова

Нам также интересно наблюдать, какое слово мы набираем. Для этого будем выводить слово в ячейку, обозначенную сейчас цифрой 3. В этом месте должны отображаться слова, введенные игроками.

Сначала нужно изменить формат ячейки и задать идентификатор:

```
   </table>
   </td><td width='56px'> 
  </td><td id='words' valign='top'>
  </td></tr>
 </table>
```

Это конец таблицы, в которой находится игровое поле.

Тройку в ячейке можно убрать.

Добавляем формирование слова при помощи функции wch в обработчике события ячеек поля:

```
function initC() {
  ob('ctbl').onclick = function(e) {
    var o = getSource(getEvent(e))
    if (!validC(o.index)) return
    if (state == INIT) {
      unsel(cuc)
      selC(o)
    } else if (state == CELL) {
      if (cuc) clear(cuc)
      o.setLetter()
      selC(o)
    } else if (state == WORD) {
      chain.push(o.index)
      selW(o)
      ob('words').innerHTML = wch(chain, getword, '')
    }
  }
  cuc = ob('c0')
  zero = getIndex('c0')
  selC = select('#fc8', 'cuc')
  selW = select('#fc8', 'cuw')
  cellProps()
}
```

Сохраним и обновим страницу.

Убедимся, что новое слово выводится.

Это временное решение, необходимое для контроля.

Далее мы определим вывод более точно.

5. Работа JS-05. Проверка слова

Цели:
- объекты JavaScript;
- взаимодействие с сервером при помощи AJAX

Задачи:
- описать объект игры;
- проверить новое слово на поле;
- проверить новое слово в словаре.

5.1. Объект игры

Нам нужно куда-то записывать слова, введенные игроком. Игроков, по определению, может быть один или двое. Обозначим их буквами А и Б, а числами как 0 и 1.

Определим объект игры как объект JavaScript:

```
<script>
var game = {
  name: ['А', 'Б'],
  usersNum: 1,
  current: 0,
  words: [],
  who: [],
  init: function(c, n) {
    this.words = [c]
    this.who = [5]
    this.usersNum = n
  },
  add: function(c, h) {
    this.words.push(c)
    this.who.push(h)
    this.current = (h + 1) % this.usersNum
  },
  state: function() {
    var i, n, w, res = ''
    for (i = 1, n = this.words.length; i < n; i++) {
      w = wch(this.words[i], getword, '').toLowerCase()
      res += '[' + this.name[this.who[i]] + ': ' + w + '] '
    }
    return res
  }
}
var IDLE=0, INIT=1, CELL=2, WORD=3, OVER=4, state=INIT
```

Обратим внимание, что элементы объекта разделены запятыми (они выделены цветом). Поле name — это условное имя игрока, usersNum — количество игроков, current — это индекс 0 или 1 текущего игрока. Поле words — введенные слова, а поле who — кто вводил.

Функция init принимает цепочку начального слова и количество игроков. Начальное слово вводит игрок номер 5.

Функция add принимает цепочку нового слова и индекс игрока, который ввел слово. Функция вычисляет индекс следующего игрока.

Функция state формирует запись о всех словах, введенных на поле к настоящему моменту. Для этого она просматривает цепочки всех слов, извлекает слово в переменную w, определяет, кто ввел слово по данным в массиве who, и формирует запись о слове в формате [A: город]. Важно, чтобы в этой записи не было простых пробелов, чтобы она не разрывалась.

Кажется, нам не хватает места, куда выводить набираемое слово.

В файле стилей определим стиль для еще одного блока:

```
.curw {
 padding: 4px 0px 4px 0px;
}
```

Сам блок разместим перед кнопками. В блоке две надписи. Одна будет показывать, чей сейчас ход, а другая — вводимое слово:

```
 </table>
 <div id='hdiv' class='hint'>Введите начальное слово</div>
 <div class='curw'>
  <span id='ct'>Ваш ход:</span> <span id='cw'> </span>
 </div>
 <table width='100%' cellspacing='0' cellpadding='0' border='0'><tr>
```

Сначала инициализируем объект игры.

Первый случай — функция window.onload:

```
window.onload = function() {
  initA()
  initC()
  initB()
  . . .
  cell(15).innerHTML = 'A'
  game.init([10, 10, 11, 12, 13, 14], 1)
  unsel(cuc)
  selC(cell(15))
  setState(CELL)
}
```

Второй случай — когда слово набирается, функция initA:

```
function initA() {
  ob('atbl').onclick = function(e) {
    . . .
    if (cuc.index == 15) {
      setState(CELL)
      cuc.setLetter()
      game.init([10, 10, 11, 12, 13, 14], 1)
    }
  }
  cua = ob('a0')
  selA = select('#fcf', 'cua')
}
```

Набираемое слово отображаем в функции initC, как и прежде:

```
    ob('cw').innerHTML = wch(chain, getword, '')
```

Нужно только изменить идентификатор объекта.

Добавляем слово в объект игры в функции onSave, там же выясняем, чей ход следующий:

```
function onSave() {
  if (state != WORD) return
  if (window.chain == undefined) return
  if (chain.length < 2) return
  if (!wch(chain, isinchy, false, chain[0])) return
  cells(unsel)
  cuc.fixLetter()
  cuc = null
  game.add(chain, game.current)
  ob('words').innerHTML = game.state()
  ob('cw').innerHTML = ' '
  var free = freeCell()
  if (free < 0) {
    setState(OVER)
    ob('ct').innerHTML = ' '
  } else {
    selC(cell(free))
    cuc.setLetter()
    if (game.current == 0) {
      ob('ct').innerHTML = 'Ваш ход:'
    } else {
      ob('ct').innerHTML = 'Ход противника:'
    }
  }
}
```

Сохраним и обновим страницу.

Тестируем, убеждаемся, что слова выводятся и формируется правильная надпись, показывающая, чей сейчас ход.

5.2. Проверка слова в списке слов

Следующая задача — убедиться, что новое слово отсутствует на поле. Сначала проще проверить, не вводилось ли уже это слово. Для этого нужно сравнить слово со всеми словами цепочек game.words.

Можно добавить в объект game функцию, которая это сделает.

Параметром функции пусть будет цепочка:

```
  find: function(c) {
    var i, n, v = wch(c, getword, '')
    for (i = 0, n = this.words.length; i < n; i++) {
      w = wch(this.words[i], getword, '')
      if (v == w) return 1
    }
    return 0
  },
```

Не забываем, что элементы объекта разделяются запятыми.

Включаем эту проверку в onSave:

```
function onSave() {
  if (state != WORD) return
  if (window.chain == undefined) return
  if (chain.length < 3) return
  if (!wch(chain, isinchy, false, chain[0])) return
  if (game.find(chain)) {
    alert('Слово есть на поле')
    return
  }
  . . .
}
```

Сохраним и обновим страницу.

Пытаемся ввести новую букву «Г» и слово «ГОРОД».

5.3. Поиск слова на поле

Искать слово на поле буем самым тупым способом, проверяя букву за буквой новое слово. Понадобится функция, которая ищет первую букву слова на поле, и рекурсивная функция, которая найдет остаток слова.

Рекурсивная функция:

```
function tail(i, w) {
  var s = w.substring(0, 1)
  var t = w.substring(1)
  for (var j = 0; j < 25; j++) {
    if (i != j) {
      if (cell(j).letter == s) {
        if (validw(j)) {
          chain.push(j)
          if (t.length > 0) {
            if (tail(j, t)) return true
            chain.pop()
          } else {
            return true
          }
        }
      }
    }
  }
  return false
}
```

Для проверки допустимости цепочки слова используется глобальная переменная chain и функция validW.

Функция ищет очередную букву слова s. Если буква найдена, то проверяем ее при помощи validW. Если проверка проходит, то проталкиваем в цепочку индекс ячейки поля, проверяем условие завершения и вызываем рекурсивно функцию для поиска оставшегося хвоста t.

Другая функция ищет первую букву, и если буква найдена, то формирует начало цепочки и вызывает рекурсивную функцию для хвоста t:

```
function findW(w) {
  var c = chain
  var r = false
  var s = w.substring(0, 1)
  var t = w.substring(1)
  for (var j = 0; j < 25; j++) {
    if (cell(j).letter == s) {
      chain = [0, j]
      r = tail(j, t)
      if (r) break
    }
  }
  chain = c
  return r
}
```

Вызываем эту функцию в onSave:

```
function onSave() {
  if (state != WORD) return
  if (window.chain == undefined) return
  if (chain.length < 3) return
  if (!wch(chain, isinchy, false, chain[0])) return
  if (game.find(chain)) {
    alert('Слово есть на поле')
    return
  }
  var w = wch(chain, getword, '')
  if (findW(w)) {
    alert('Слово есть на поле')
    return
  }
  . . .
}
```

Сохраним и обновим страницу.

Убеждаемся, что нельзя ввести слово, которое есть на поле.

Для продолжения нужно реализовать отмену, функция onBack. По нашему сценарию, отменить можно только режим WORD. Отменяем выделение ячеек, убираем набранный текст, выделяем введенную букву и возвращаем состояние CELL:

```
function onBack() {
  if (state != WORD) return
  cells(unsel)
  ob('cw').innerHTML = ' '
  selC(cuc)
  setState(CELL)
}
```

Теперь опишем функцию, которая выведет сообщение об ошибке в строку подсказки, вместо того, чтобы показывать пользователю alert:

```
function werror(a, b) {
  hint('<span style="color:red">' + a + '</span>. ' + b)
}
```

Функция onSave. Здесь несколько изменений:

```
function onSave() {
  if (state != WORD) return
  if (window.chain == undefined) return
  if (chain.length < 3) {
    werror('Слово не может иметь одну букву', 'Нажмите «Отмена»')
    return
  }
  if (!wch(chain, isinchy, false, chain[0])) {
    werror('Новая буква должна входить в слово', 'Нажмите «Отмена»')
    return
  }
  if (game.find(chain)) {
    werror('Слово есть на поле', 'Нажмите «Отмена»')
    return
  }
  var w = wch(chain, getword, '')
  if (findW(w)) {
    werror('Слово есть на поле', 'Нажмите «Отмена»')
    return
  }
  . . .
}
```

Сохраним и обновим страницу.

Убеждаемся, что все проверки сообщают об ошибках.

5.4. Поиск слова в словаре

Остается проверить слово в словаре. Принцип проверки следующий. Сценарий проверки стороны сервера возвращает строку "0", если слово не обнаружено. Будем контролировать только этот ноль. В любом другом случае будем считать, что слово прошло проверку. Следующая функция проверки использует try для непредвиденной ситуации:

```
function checkW(w) {
  var u = encodeURIComponent(w)
  var r = 'http://revol.ponocom.ru/ksg/spell.php?w=' + u
  var xhr = new XMLHttpRequest()
  if (!xhr) return 1
  try {
    xhr.open('GET', r, false)
    xhr.send()
    if (xhr.status == 200) {
      if (xhr.responseText == '0') return 0
      alert('Success')
      return 1
    }
  } catch(e) {
    return 1
  }
  return 1
}
```

Для проверки используем взаимодействие с сервером посредством запроса AJAX. Асинхронный запрос значительно усложнит алгоритмы, используем синхронный запрос. Если он выполнится, то достаточно быстро, все равно никакой другой работы в этот момент у игрока нет. Чтобы игроку не было в момент проверки скучно, нужно бы в строку подсказки вывести индикатор выполнения, например, непрерывно бегущие точки.

Вызываем проверку слова в onSave:

```
function onSave() {
  . . .
  var w = wch(chain, getword, '')
  if (findW(w)) {
    werror('Слово есть на поле', 'Нажмите «Отмена»')
    return
  }
  if (!checkW(w)) {
    werror('Слово отсутствует в словаре', 'Нажмите «Отмена»')
    return
  }
  . . .
}
```

Сохраним и обновим страницу.

Убедимся, что проверка работает, удалим alert из функции checkW.

Проверка нужна также, когда вводится начальное слово после нажатия кнопки «Новая». Изменяем момент окончания ввода начального слова.

Функция initA:

```
function initA() {
  ob('atbl').onclick = function(e) {
    if (state != CELL && state != INIT) return
    unsel(cua)
    selA(getSource(getEvent(e)))
    cuc.setLetter()
    if (state != INIT) return
    cuc.fixLetter()
    unsel(cuc)
    selC(cell(cuc.index + 1))
    if (cuc.index == 15) {
      chain = [10, 10, 11, 12, 13, 14]
      var w = wch(chain, getword, '')
      if (!checkW(w)) {
        werror('Слово отсутствует в словаре', 'Нажмите «Новая»')
        return
      }
      setState(CELL)
      cuc.setLetter()
      game.init(chain, 1)
    }
  }
  cua = ob('a0')
  selA = select('#fcf', 'cua')
}
```

Сохраним и обновим страницу.

Нажимаем кнопку «Новая» и вводим «АБВГД».

5.5. Счет игры

Нужно также вычислять баллы игроков и выводить их, например, после слов. В объекте игры определим функцию, которая вычислит баллы, и вызовем ее в функции state объекта игры:

```
  result: function() {
    var balls = [0,0], i, n, w
    for (i = 1, n = this.words.length; i < n; i++) {
      w = wch(this.words[i], getword, '')
      balls[this.who[i]] += w.length
    }
    return balls
  },
  state: function() {
    var i, n, w, res = ''
    for (i = 1, n = this.words.length; i < n; i++) {
      w = wch(this.words[i], getword, '').toLowerCase()
      res += '[' + this.name[this.who[i]] + ': ' + w + '] '
    }
    return res + '[' + this.result() + ']'
  }
```

Не забываем про запятую, которая разделяет поля объекта.

Чтобы протестировать, установим при вводе начального слова двух игроков, функция initA:

```
function initA() {
  ob('atbl').onclick = function(e) {
    . . .
    if (cuc.index == 15) {
      chain = [10, 10, 11, 12, 13, 14]
      var w = wch(chain, getword, '')
      if (!checkW(w)) {
        werror('Слово отсутствует в словаре', 'Нажмите «Новая»')
        return
      }
      setState(CELL)
      cuc.setLetter()
      game.init(chain, 2)
    }
  }
  cua = ob('a0')
  selA = select('#fcf', 'cua')
}
```

Сохраним и обновим страницу.

Убедимся, что баллы игроков выводятся.

Теперь нужно очищать выведенные слова, когда инициализируется новая игра. Функция onGame:

```
function onGame() {
  ob('words').innerHTML = ' '
  cells(clear)
  unsel(cua)
  selA(ob('a0'))
  selC(cell(10))
  setState(INIT)
}
```

5.6. Асинхронный запрос AJAX

Браузерам не нравится синхронный запрос AJAX, потому что он тормозит выполнение потока, который в браузере один. Поэтому нужно перейти к асинхронному запросу.

Для этого нужно определить четыре функции, по две для каждой проверки. Функция suci соответствует действиям функции initA в случае, если слово проходит проверку. Функция erri соответствует действиям функции initA в случае, если слово не проходит проверку. Аналогично две функции sucw и errw соответствуют действиям функции onSave.

Тогда функция асинхронного запроса примет вид:

```
function checkW(w, sucf, errf) {
  var u = encodeURIComponent(w)
  var r = 'http://revol.ponocom.ru/ksg/spell.php?w=' + u
  var xhr = new XMLHttpRequest()
  if (!xhr) sucf()
  xhr.onerror = sucf
  xhr.onload = function() {
    if (xhr.status == 200) {
      if (xhr.responseText == '0') errf(); else sucf();
    } else {
      sucf()
    }
  }
  xhr.open('GET', r, true)
  xhr.send()
}
```

Вызываем эту функцию в конце функции onSave с параметрами w, errw и sucw, и в конце функции initA с параметрами w, erri и suci.

Для примера, функция errw:

```
function errw {
  werror('Слово отсутствует в словаре', 'Нажмите «Отмена»')
}
```

На этом работы по JavaScript завершены.