Содержание

1 Текст задания

Задание 10. Написать программу, играющую в игру «Борьба за жизнь».

2 Правила игры

Два игрока имеют по N шашек и играют в поле MxM клеток, делая ходы по очереди.

Возможные ходы:

1) переставить свою шашку на соседние свободные клетки

2) переставить свою шашку через поле, если оно занято другой шашкой.

Если шашка с трёх сторон окружена шашками противника, она снимается с поля. Если пустая клетка с 3-х сторон окружена шашками одного игрока, на её место ставится новая шашка этого игрока. Цель игры – убрать с доски все шашки противника.

3 Текст программы с комментариями

index.html

```
<!DOCTYPE html>

<html lang="ru">

    <head>

        <title>Игра "Борьба за жизнь" - Курсовая работа</title>

        <meta charset="UTF-8">

        <meta name="viewport" content="width=device-width, initial-
scale=1.0">

        <link rel="stylesheet" type="text/css" href="css/style.css">

        <script type="text/javascript" src="lib/jquery-
2.2.4.min.js"></script>

        <script type="text/javascript" src="js/strlife.js"></script>

    </head>

    <body>

        <div class="wrapper">

            <div id="condition">

                <h2>Правила игры:</h2>

                <p>Два игрока имеют по N шашек и играют в поле MxM клеток,
делая ходы по очереди.</p>

                <p>Возможные ходы:</p>

                <p>1) переставить свою шашку на соседние свободные клетки</p>

                <p>2) переставить свою шашку через поле, если оно занято
другой шашкой.</p>

                <p>Если шашка с трёх сторон окружена шашками противника, она
снимается с поля. Если пустая клетка с 3-х сторон окружена шашками одного
игрока, на её место ставится новая шашка этого игрока. Цель игры – убрать с
доски все шашки противника.</p>

            </div>

            <div id="input">

                <h2>Входные параметры</h2>

                <p><label>Размер игрового поля (M): <input id="sizeBoard"
type="number" value="6"></label></p>

                <p><label>Число шашек у каждого из игроков (N): <input
id="countCheckers" type="number" value="3"></label></p>
```

```
            </div>
            <div id="output">
                <h2>Выходные параметры</h2>
                <p id="message" class="center message"></p>
                <div class="chess"></div>
            </div>
        </div>
    </body>
</html>
```

style.css

```
@font-face
{
  font-family:Roboto;
  src:url(fonts/roboto/Roboto.woff2) format(woff2),
url(fonts/roboto/Roboto.woff) format(woff), url(fonts/roboto/Roboto.ttf)
format(truetype);
  font-weight:400;
  font-style:normal;
}

*
{
  margin:0;
  padding:0;
}

html
{
  font-size:62.5%;
}
```

```
body
{
  font-family:Roboto;
}

h2
{
  font-size:2em;
  background-color:#39c;
  color:#fff;
  text-align:center;
  padding:5px 8px;
}

p
{
  font-size:1.6em;
  margin:5px;
}

input
{
  font-size:1em;
}

label
{
  display:block;
  width:100%;
  text-align:center;
}
```

```
label > input
{
  width:100%;
  box-sizing:border-box;
  text-align:center;
  outline:0;
  padding:4px 2px;
}

#condition p
{
  text-align:justify;
}

#output
{
  border-bottom:1px solid #39c;
}

.center
{
  text-align:center;
}

.message
{
  margin-top:25px;
  font-size:1.8em;
}

.wrapper
{
  max-width:800px;
```

```
  width:calc(100%-20px);
  border-left:1px solid #39c;
  border-right:1px solid #39c;
  margin:0 auto;
}

.chess
{
  font-size:1.6em;
  font-weight:700;
  text-transform:uppercase;
  padding:40px;
}

.chess__wrapper
{
  position:relative;
  border:3px solid #7c6249;
}

.chess__signs-numbers,.chess__signs-chars
{
  position:absolute;
  -ms-flex-pack:distribute;
  justify-content:space-around;
  display:flex;
}

.chess__signs-numbers
{
  -webkit-box-orient:vertical;
  -webkit-box-direction:reverse;
  -ms-flex-direction:column-reverse;
```

```
  flex-direction:column-reverse;
  top:0;
  bottom:0;
}

.chess__signs-numbers--left
{
  left:-25px;
}

.chess__signs-numbers--right
{
  right:-25px;
}

.chess__signs-chars
{
  left:0;
  right:0;
}

.chess__signs-chars--top
{
  top:-30px;
}

.chess__signs-chars--bottom
{
  bottom:-30px;
}

.chess__board-row
{
```

```
  display:flex;
}

.chess__board-element
{
  background-color:#7c6249;
  width:100%;
}

.chess__board-row:nth-of-type(2n) .chess__board-element:nth-of-
type(2n),.chess__board-row:nth-of-type(2n+1) .chess__board-element:nth-of-
type(2n+1)
{
  background-color:#e7cfa9;
}

.chess__board-element-wrapper
{
  position:relative;
  width:100%;
  height:100%;
  padding-top:100%;
  border:2px solid transparent;
}

.chess__board-element-wrapper.checker
{
  cursor:pointer;
  border:2px solid #fff;
  box-sizing:border-box;
  border-radius:50%;
  padding-top:84%;
  width:84%;
  height:84%;
```

```
  margin:8% auto;
}

.chess__board-element-wrapper.checker::after
{
  position:absolute;
  top:0;
  bottom:0;
  left:0;
  right:0;
}

.chess__board-element-wrapper.road
{
  cursor:pointer;
  border:2px solid #fff;
  box-sizing:border-box;
  border-radius:50%;
  width:84%;
  padding-top:84%;
  height:84%;
  margin:8% auto;
}

.chess__board-element-wrapper.road:hover
{
  background:rgba(255,255,255,0.2);
}

.chess__board-element-wrapper.checker.white_checker
{
  background:radial-gradient(circleclosest-side,#d6d7cf20%,#f0f0f0100%);
  border:2px solid #000;
```

```
}

.chess__board-element-wrapper.checker.black_checker
{
  background:radial-gradient(circleclosest-side,#5555551%,#33333370%);
  border:2px solid #fff;
}

.chess__board-element-wrapper.checker.white_checker:not(.selected):hover
{
  background:radial-gradient(circleclosest-side,#d6d7cf20%,#f0f0f080%);
}

.chess__board-element-wrapper.checker.black_checker:not(.selected):hover
{
  background:radial-gradient(circleclosest-side,#5555551%,#33333350%);
}

.chess__board-element-wrapper.checker.white_checker.selected,.chess__board-
element-wrapper.checker.black_checker.selected
{
  border:2px solid #c00;
}

@media only screen and max-width 720px {
  .chess
  {
    font-size:1.4em;
    padding:30px;
  }

  .chess__signs-numbers--left
  {
```

```
    left:-22px;
  }

  .chess__signs-numbers--right
  {
    right:-22px;
  }

  .chess__signs-chars--top
  {
    top:-25px;
  }

  .chess__signs-chars--bottom
  {
    bottom:-25px;
  }
}

@media only screen and max-width 480px {
  .chess
  {
    font-size:1.2em;
    padding:20px;
  }

  .chess__signs-numbers--left
  {
    left:-16px;
  }

  .chess__signs-numbers--right
  {
```

```
    right:-16px;
  }

  .chess__signs-chars--top
  {
    top:-20px;
  }

  .chess__signs-chars--bottom
  {
    bottom:-20px;
  }
}
```

strlife.js

```
/**
 * Проверка значения на int
 *
 * @param {type} val
 *   Проверяемое значение
 * @returns {boolean}
 *   Результат проверки
 */
function isInt(val)
{
    var result = false;
    if (Math.floor(val).toString() === val && jQuery.isNumeric(val))
    {
        result = true;
    }
    return result;
}
```

```
/**
 * Является ли число положительным
 *
 * @param {type} val
 *   Проверяемое значение
 * @returns {boolean}
 *   Результат проверки
 */
function numPositive(val)
{
    var result = false;
    if (val >= 0)
    {
        result = true;
    }
    return result;
}


/**
 * Конвертирует строковые значения массива в целочисленные
 *
 * @param {array} arr
 *   Входной массив
 * @returns {array}
 *   Целочисленный массив
 */
function convertArrayToInt(arr)
{
    if (Array.isArray(arr))
    {
        var output = arr.map(function(item)
        {
            return parseInt(item);
```

```
        });
    }
    else
    {
        output = arr;
    }
    return output;
}

/**
 * Возвращает значение атрибута value
 *
 * @param {string} selector
 *   Селектор
 * @returns {string}
 *   Значение элемента формы
 */
function getVal(selector)
{
    return jQuery(selector).val();
}

/**
 * Возвращает размер игрового поля
 *
 * @returns {string}
 *   Размер игрового поля
 */
function getSizeBoard()
{
    return getVal('#sizeBoard');
}
```

```
/**
 * Возвращает количество шашек для каждого игрока
 *
 * @returns {string}
 *   Количество шашек игрока
 */
function getCountCheckers()
{
    return getVal('#countCheckers');
}

/**
 * Возвращает допустимый алфавит для игровой доски
 *
 * @returns {string}
 *   Допустимый алфавит
 */
function getAlphabet()
{
    var alphabet = 'ABCDEFGHI';
    return alphabet;
}

/**
 * Возвращает максимальный размер игрового поля
 *
 * @returns {int}
 *   Максимальная размерность игрового поля
 */
function getMaxSizeBoard()
{
    var alphabet = getAlphabet();
    return alphabet.length;
```

```
}

/**
 * Проверяет размер игрового поля с максимально допустимым значением
 *
 * @param {int} size
 *   Размер игрового поля
 * @returns {boolean}
 *   Результат проверки
 */
function checkMaxSizeBoard(size)
{
    size = parseInt(size);
    var result = true;
    var maxsize = getMaxSizeBoard();
    if (size > maxsize)
    {
        result = false;
    }
    return result;
}

/**
 * Проверяет количество шашек игрока с максимально допустимым значением
 *
 * @param {int} size
 *   Размер игрового поля
 * @param {int} count
 *   Количество шашек игрока
 * @returns {boolean}
 *   Результат проверки
 */
function checkMaxCountCheckers(size, count)
```

```
{
    size = parseInt(size);
    count = parseInt(count);
    var result = true;
    var dim = size * size; // Количество ячеек на игровом поле MxM
    var maxcount;
    if (dim % 2 === 0)
    {
        maxcount = (dim - size * 2) / 2;
    }
    else
    {
        maxcount = (dim - size) / 2;
    }
    if (count > maxcount)
    {
        result = false;
    }
    return result;
}

/**
 * Реализует игровое поле
 *
 * @param {string} container
 *   Селектор контейнера для игрового поля
 * @param {int} dim
 *   Размер игрового поля
 * @param {int} cnt
 *   Количество шашек игрока
 * @returns {object}
 *   Игровое поле
 */
```

```
function Board(container, dim, cnt)
{
    var self = this; // Текущий объект
    self.container = container; // Селектор контейнера для игрового поля
    self.white_checker = 'white_checker'; // Класс белых шашек
    self.black_checker = 'black_checker'; // Класс черных шашек
    self.this_checker = ''; // Класс шашки текущего игрока
    self.selected = 'selected'; // Класс выбранной шашки
    self.size = parseInt(dim); // Размер игрового поля
    self.count = parseInt(cnt); // Количество шашек игрока
    self.statesBoard = []; // Состояния ячеек игровой доски
    self.selectChecker = null; // Выбранная шашка
    self.sideChecker = null; // Выбранная сторона (кому принадлежит шашка)
    self.gameEnd = false; // Игра завершена
    self.winner = null; // Победитель

    /**
     * Возвращает размер игрового поля
     *
     * @returns {int}
     */
    self.getSize = function()
    {
        return self.size;
    };

    /**
     * Возвращает количество шашек для одного игрока
     *
     * @returns {int}
     */
    self.getCount = function()
    {
```

```
    return self.count;
};

/**
 * Запоминает состояния игровой доски
 *
 * @param {array} statesBoard
 *    Состояния игровой доски
 */
self.setStatesBoard = function(statesBoard)
{
    self.statesBoard = statesBoard;
};

/**
 * Получает состояния игровой доски
 */
self.getStatesBoard = function()
{
    return self.statesBoard;
};

/**
 * Запоминает координаты выбранной шашки
 *
 * @param {array} indexes
 *    Координаты выбранной шашки
 */
self.setSelectChecker = function(indexes)
{
    self.selectChecker = indexes;
};
```

```
/**
 * Получает координаты выбранной шашки
 */
self.getSelectChecker = function()
{
    return self.selectChecker;
};

/**
 * Запоминает сторону выбранной шашки
 *
 * @param {array} classname
 *   Класс выбранной стороны
 */
self.setSideChecker = function(classname)
{
    self.sideChecker = classname;
};

/**
 * Получает сторону выбранной шашки
 */
self.getSideChecker = function()
{
    return self.sideChecker;
};

/**
 * Получает противоположную сторону выбранной шашки
 */
self.getSideCheckerRival = function()
{
    var sideCheckerRival = 0;
```

```
    if (self.sideChecker === 1)
    {
        sideCheckerRival = 2;
    }
    else if (self.sideChecker === 2)
    {
        sideCheckerRival = 1;
    }
    return sideCheckerRival;
};

/**
 * Смена хода для другого игрока
 */
self.changeSideChecker = function()
{
    self.setSideChecker(self.getSideCheckerRival());
};

/**
 * Создает белые шашки в массиве состояний ячеек
 */
self.createWhiteCheckers = function()
{
    var statesBoard = self.getStatesBoard();
    var size = self.getSize();
    var cnt = self.getCount();
    for (var i = size - 1; i >= 0; i--)
    {
        for (var j = size - 1; j >= 0; j--)
        {
            if (cnt > 0)
            {
```

```
                statesBoard[i][j] = 1;
                cnt--;
            }
        }
    }
    self.setStatesBoard(statesBoard);
};

/**
 * Создает черные шашки в массиве состояний ячеек
 */
self.createBlackCheckers = function()
{
    var statesBoard = self.getStatesBoard();
    var size = self.getSize();
    var cnt = self.getCount();
    for (var i = 0; i < size; i++)
    {
        for (var j = 0; j < size; j++)
        {
            if (cnt > 0)
            {
                statesBoard[i][j] = 2;
                cnt--;
            }
        }
    }
    self.setStatesBoard(statesBoard);
};

/**
 * Инициализация состояний ячеек игровой доски
 * Представляет из себя двумерный массив, где:
```

```
 * 0 - ячейка пустая
 * 1 - ячейка занята первым (белым) игроком
 * 2 - ячейка занята вторым (черным) игроком
 */
self.initStatesBoard = function()
{
    var statesBoard = self.getStatesBoard();
    var size = self.getSize();
    for (var i = 0; i < size; i++)
    {
        statesBoard[i] = [];
        for (var j = 0; j < size; j++)
        {
            statesBoard[i][j] = 0;
        }
    }
    self.setStatesBoard(statesBoard);
    self.createWhiteCheckers();
    self.createBlackCheckers();
    // Проверка окончания игры
    self.gameOver();
    // Сообщение пользователю
    self.showMessage();

};

/**
 * Формирует тег div с указанным классом
 *
 * @param {string} classname
 *   Класс
 * @returns {string}
 *   Сформированный тег
```

```
 */
self.createDivByClass = function(classname)
{
    var result = '<div class="' + classname + '"></div>';
    return result;
};

/**
 * Добавляет содержимое рядом блочным элементом
 *
 * @param {string} classparent
 *   Класс родительского элемента
 * @param {string} html
 *   Добавляемый контент
 */
self.htmlAppend = function(classparent, html)
{
    classparent = classparent.replace(' ', '.');
    jQuery('.' + classparent).append(html);
};

/**
 * Заменяет содержимое в блочном элементе
 *
 * @param {string} classparent
 *   Класс родительского элемента
 * @param {string} html
 *   Добавляемый контент
 */
self.htmlReplace = function(classparent, html)
{
    classparent = classparent.replace(' ', '.');
    jQuery('.' + classparent).html(html);
```

```
};

/**
 * Заменяет содержимое в группе блочных элементов
 *
 * @param {string} classname
 *   Класс родительских элементов
 * @param {string} html
 *   Добавляемый контент
 */
self.htmlReplaceEach = function(classname, html)
{
    jQuery('.' + classname).each(function()
    {
        jQuery(this).html(html);
    });
};

/**
 * Отрисовка игрового поля
 */
self.drawBoard = function()
{
    // Инициализируем состояния ячеек игровой доски
    self.initStatesBoard();
    var size = self.getSize();

    // Обертка игрового поля
    var chess_wrapper = 'chess__wrapper';
    var chess_wrapper_div = self.createDivByClass(chess_wrapper);
    self.htmlReplace(container, chess_wrapper_div);

    // Игровое поле
```

```
        var chess_board = 'chess__board';
        var chess_board_div = self.createDivByClass(chess_board);
        self.htmlReplace(chess_wrapper, chess_board_div);

        // Строки ячеек
        var chess_board_row = 'chess__board-row';
        var chess_board_row_div = '';
        for (var i = 0; i < size; i++)
        {
            chess_board_row_div += self.createDivByClass(chess_board_row);
        }
        self.htmlReplace(chess_board, chess_board_row_div);

        // Ячейки
        var chess_board_element = 'chess__board-element';
        var chess_board_element_div = '';
        for (var i = 0; i < size; i++)
        {
            chess_board_element_div                                    +=
self.createDivByClass(chess_board_element);
        }
        self.htmlReplaceEach(chess_board_row, chess_board_element_div);
        // Место для шашек в ячейках
        var chess_board_element_wrapper = 'chess__board-element-wrapper';
        var                  chess_board_element_wrapper_div                =
self.createDivByClass(chess_board_element_wrapper);
        self.htmlReplaceEach(chess_board_element,
chess_board_element_wrapper_div);
        // Обертка координат ячеек
        var chess_signs = 'chess__signs';
        var chess_signs_div = self.createDivByClass(chess_signs);
        self.htmlAppend(chess_wrapper, chess_signs_div);
```

```
        // Подписи числовых координат ячеек
        var signs_numbers = 'chess__signs-numbers chess__signs-numbers--';
        // Подписи символьних координат ячеек
        var signs_chars = 'chess__signs-chars chess__signs-chars--';
        // Подписи координат ячеек обертка слева
        var signs_numbers_left = signs_numbers + 'left';
        var                      signs_numbers_left_div                      =
self.createDivByClass(signs_numbers_left);
        self.htmlAppend(chess_signs, signs_numbers_left_div);
        // Подписи координат ячеек обертка справа
        var signs_numbers_right = signs_numbers + 'right';
        var                     signs_numbers_right_div                      =
self.createDivByClass(signs_numbers_right);
        self.htmlAppend(chess_signs, signs_numbers_right_div);
        // Подписи координат ячеек обертка сверху
        var signs_chars_top = signs_chars + 'top';
        var signs_chars_top_div = self.createDivByClass(signs_chars_top);
        self.htmlAppend(chess_signs, signs_chars_top_div);
        // Подписи координат ячеек обертка снизу
        var signs_chars_bottom = signs_chars + 'bottom';
        var                      signs_chars_bottom_div                      =
self.createDivByClass(signs_chars_bottom);
        self.htmlAppend(chess_signs, signs_chars_bottom_div);
        // Числовая нумерация
        var numeric_numbering = '';
        for (var i = 0; i < size; i++)
        {
            numeric_numbering += '<span>' + (i + 1) + '</span>';
        }
        // Подписи координат ячеек слева
        self.htmlReplace(signs_numbers_left, numeric_numbering);
        // Подписи координат ячеек справа
        self.htmlReplace(signs_numbers_right, numeric_numbering);
        // Символьная нумерация
```

```
    var symbolic_numbering = '';
    var alphabet = getAlphabet();
    for (var i = 0; i < size && size <= alphabet.length; i++)
    {
        symbolic_numbering += '<span>' + alphabet.charAt(i) + '</span>';
    }
    // Подписи координат ячеек сверху
    self.htmlReplace(signs_chars_top, symbolic_numbering);
    // Подписи координат ячеек снизу
    self.htmlReplace(signs_chars_bottom, symbolic_numbering);
};

/**
 * Отображает возможный переход
 *
 * @param {int} i
 *   Координата строки
 * @param {int} j
 *   Координата столбца
 */
self.drawRoad = function(i, j)
{
    jQuery('#coord-' + i + '-' + j).addClass('road');
};

/**
 * Сбрасывавает возможные ходы для шашек
 */
self.clearRoad = function()
{
    jQuery('.road').removeClass('road');
};
```

```
    /**
     * Поиск возможных ходов для шашки
     *
     * @param {int} i
     *   Строка в массиве состояний
     * @param {int} j
     *   Столбец в массиве состояний
     * @returns {int} roads
     *   Количество возможных ходов
     */
    self.findRoads = function(i, j)
    {
        var roads = 0;
        var size = self.getSize();
        var statesBoard = self.getStatesBoard();
        self.clearRoad(); // Стираем предыдущие пути
        // Не вышли за пределы игровой доски
        if (i >= 0 && i < size && j >= 0 && j < size)
        {
            // Является шашкой одного из игроков
            if (statesBoard[i][j] === 1 || statesBoard[i][j] === 2)
            {
                // - - -
                // Перестановка шашки на одну из соседних свободных клеток
                // Перестановка шашки через ячейку, если оно занято другой
шашкой
                // - - -
                // Проверяем клетку сверху
                if (i - 1 >= 0)
                {
                    if (statesBoard[i - 1][j] === 0)
                    {
                        self.drawRoad(i - 1, j);
```

```
                        roads++;
                    }
                    else if (i - 2 >= 0)
                    {
                        if (statesBoard[i - 2][j] !== 1 && statesBoard[i -
2][j] !== 2)
                        {
                            self.drawRoad(i - 2, j);
                            roads++;
                        }
                    }
                }
                // Проверяем клетку снизу
                if (i + 1 < size)
                {
                    if (statesBoard[i + 1][j] === 0)
                    {
                        self.drawRoad(i + 1, j);
                        roads++;
                    }
                    else if (i + 2 < size)
                    {
                        if (statesBoard[i + 2][j] !== 1 && statesBoard[i +
2][j] !== 2)
                        {
                            self.drawRoad(i + 2, j);
                            roads++;
                        }
                    }
                }
                // Проверяем клетку слева
                if (j - 1 >= 0)
                {
                    if (statesBoard[i][j - 1] === 0)
```

```
                    {
                        self.drawRoad(i, j - 1);
                        roads++;
                    }
                    else if (j - 2 >= 0)
                    {
                        if (statesBoard[i][j - 2] !== 1 && statesBoard[i][j -
2] !== 2)
                        {
                            self.drawRoad(i, j - 2);
                            roads++;
                        }
                    }
                }
                // Проверяем клетку справа
                if (j + 1 < size)
                {
                    if (statesBoard[i][j + 1] === 0)
                    {
                        self.drawRoad(i, j + 1);
                        roads++;
                    }
                    else if (j + 2 < size)
                    {
                        if (statesBoard[i][j + 2] !== 1 && statesBoard[i][j +
2] !== 2)
                        {
                            self.drawRoad(i, j + 2);
                            roads++;
                        }
                    }
                }
                // Проверяем клетку сверху слева
                if (i - 1 >= 0 && j - 1 >= 0)
```

```
                {
                    if (statesBoard[i - 1][j - 1] === 0)
                    {
                        self.drawRoad(i - 1, j - 1);
                        roads++;
                    }
                    else if (i - 2 >= 0 && j - 2 >= 0)
                    {
                        if (statesBoard[i - 2][j - 2] !== 1 && statesBoard[i -
2][j - 2] !== 2)
                        {
                            self.drawRoad(i - 2, j - 2);
                            roads++;
                        }
                    }
                }
                // Проверяем клетку сверху справа
                if (i - 1 >= 0 && j + 1 < size)
                {
                    if (statesBoard[i - 1][j + 1] === 0)
                    {
                        self.drawRoad(i - 1, j + 1);
                        roads++;
                    }
                    else if (i - 2 >= 0 && j + 2 < size)
                    {
                        if (statesBoard[i - 2][j + 2] !== 1 && statesBoard[i -
2][j + 2] !== 2)
                        {
                            self.drawRoad(i - 2, j + 2);
                            roads++;
                        }
                    }
                }
```

```
                // Проверяем клетку снизу слева
                if (i + 1 < size && j - 1 >= 0)
                {
                    if (statesBoard[i + 1][j - 1] === 0)
                    {
                        self.drawRoad(i + 1, j - 1);
                        roads++;
                    }
                    else if (i + 2 < size && j - 2 >= 0)
                    {
                        if (statesBoard[i + 2][j - 2] !== 1 && statesBoard[i +
2][j - 2] !== 2)
                        {
                            self.drawRoad(i + 2, j - 2);
                            roads++;
                        }
                    }
                }
                // Проверяем клетку снизу справа
                if (i + 1 < size && j + 1 < size)
                {
                    if (statesBoard[i + 1][j + 1] === 0)
                    {
                        self.drawRoad(i + 1, j + 1);
                        roads++;
                    }
                    else if (i + 2 < size && j + 2 < size)
                    {
                        if (statesBoard[i + 2][j + 2] !== 1 && statesBoard[i +
2][j + 2] !== 2)
                        {
                            self.drawRoad(i + 2, j + 2);
                            roads++;
                        }
```

```
                    }
                }
            }
        }
        // Отслеживаем клик по возможному пути для шашки
        self.eventClickRoad();
        return roads;
    };

    /**
     * Удаляет шашку соперника с поля, если она окружена с трех сторон шашками текущего игрока
     */
    self.removeChecker = function()
    {
        var sideChecker = self.getSideChecker(); // Текущая шашка
        var sideCheckerRival = self.getSideCheckerRival(); // Шашка соперника
        var statesBoard = self.getStatesBoard();
        var newStatesBoard = statesBoard.map(function(arr)
        {
            return arr.slice();
        });
        var size = self.getSize();
        for (var i = 0; i < size; i++)
        {
            for (var j = 0; j < size; j++)
            {
                var matches = 0; // Соседних шашек соперника
                // Если проверяемая клетка соперника
                if (statesBoard[i][j] === sideCheckerRival)
                {
                    // Проверяем клетку сверху
                    if (i - 1 >= 0 && statesBoard[i - 1][j] === sideChecker)
```

```
                    {
                        matches++;
                    }
                    // Проверяем клетку снизу
                    if (i + 1 < size && statesBoard[i + 1][j] === sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку слева
                    if (j - 1 >= 0 && statesBoard[i][j - 1] === sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку справа
                    if (j + 1 < size && statesBoard[i][j + 1] === sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку сверху слева
                    if (i - 1 >= 0 && j - 1 >= 0 && statesBoard[i - 1][j - 1]
=== sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку сверху справа
                    if (i - 1 >= 0 && j + 1 < size && statesBoard[i - 1][j +
1] === sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку снизу слева
                    if (i + 1 < size && j - 1 >= 0 && statesBoard[i + 1][j -
1] === sideChecker)
                    {
```

```
                        matches++;
                    }
                    // Проверяем клетку снизу справа
                    if (i + 1 < size && j + 1 < size && statesBoard[i + 1][j +
1] === sideChecker)
                    {
                        matches++;
                    }
                    // Удаление шашки, если ее окружили три шашки соперника
                    if (matches >= 3)
                    {
                        newStatesBoard[i][j] = 0;
                    }
                }
                if (statesBoard[i][j] === sideChecker)
                {
                    // Проверяем клетку сверху
                    if (i - 1 >= 0 && statesBoard[i - 1][j] ===
sideCheckerRival)
                    {
                        matches++;
                    }
                    // Проверяем клетку снизу
                    if (i + 1 < size && statesBoard[i + 1][j] ===
sideCheckerRival)
                    {
                        matches++;
                    }
                    // Проверяем клетку слева
                    if (j - 1 >= 0 && statesBoard[i][j - 1] ===
sideCheckerRival)
                    {
                        matches++;
                    }
```

```
                    // Проверяем клетку справа
                    if (j + 1 < size && statesBoard[i][j + 1] ===
sideCheckerRival)
                    {
                        matches++;
                    }
                    // Проверяем клетку сверху слева
                    if (i - 1 >= 0 && j - 1 >= 0 && statesBoard[i - 1][j - 1]
=== sideCheckerRival)
                    {
                        matches++;
                    }
                    // Проверяем клетку сверху справа
                    if (i - 1 >= 0 && j + 1 < size && statesBoard[i - 1][j +
1] === sideCheckerRival)
                    {
                        matches++;
                    }
                    // Проверяем клетку снизу слева
                    if (i + 1 < size && j - 1 >= 0 && statesBoard[i + 1][j -
1] === sideCheckerRival)
                    {
                        matches++;
                    }
                    // Проверяем клетку снизу справа
                    if (i + 1 < size && j + 1 < size && statesBoard[i + 1][j +
1] === sideCheckerRival)
                    {
                        matches++;
                    }
                    // Удаление шашки, если ее окружили три шашки соперника
                    if (matches >= 3)
                    {
                        newStatesBoard[i][j] = 0;
                    }
```

```
            }
        }
    }
    self.setStatesBoard(newStatesBoard);
};

/**
 * Добавляет шашку на пустую клетку, если она окружена с трех сторон
 * шашками одного игрока
 */
self.addChecker = function()
{
    var sideChecker = self.getSideChecker(); // Текущая шашка
    var statesBoard = self.getStatesBoard();
    var newStatesBoard = statesBoard.map(function(arr)
    {
        return arr.slice();
    });
    var size = self.getSize();
    for (var i = 0; i < size; i++)
    {
        for (var j = 0; j < size; j++)
        {
            var matches = 0; // Соседних шашек текущего игрока
            // Если проверяемая клетка пустая
            if (statesBoard[i][j] === 0)
            {
                // Проверяем клетку сверху
                if (i - 1 >= 0 && statesBoard[i - 1][j] === sideChecker)
                {
                    matches++;
                }
                // Проверяем клетку снизу
```

```
                    if (i + 1 < size && statesBoard[i + 1][j] === sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку слева
                    if (j - 1 >= 0 && statesBoard[i][j - 1] === sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку справа
                    if (j + 1 < size && statesBoard[i][j + 1] === sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку сверху слева
                    if (i - 1 >= 0 && j - 1 >= 0 && statesBoard[i - 1][j - 1]
=== sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку сверху справа
                    if (i - 1 >= 0 && j + 1 < size && statesBoard[i - 1][j +
1] === sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку снизу слева
                    if (i + 1 < size && j - 1 >= 0 && statesBoard[i + 1][j -
1] === sideChecker)
                    {
                        matches++;
                    }
                    // Проверяем клетку снизу справа
```

```
                    if (i + 1 < size && j + 1 < size && statesBoard[i + 1][j +
1] === sideChecker)
                    {
                        matches++;
                    }
                    // Добавление новой шашки, если требуется
                    if (matches >= 3)
                    {
                        newStatesBoard[i][j] = sideChecker;
                    }
                }
            }
        }
        self.setStatesBoard(newStatesBoard);
    };

    /**
     * Отрисовка шашек на игровом поле
     */
    self.drawCheckers = function()
    {

        var statesBoard = self.getStatesBoard();
        var rows = '.chess__board-row';
        var cols = '.chess__board-element';
        self.clearRoad(); // Стираем предыдущие пути
        var i = 0;
        jQuery(rows).each(function()
        {
            var j = 0;
            jQuery(cols, this).each(function()
            {
                // Нумерация ячеек в виде одномерного массива
```

```
                jQuery(this).children().attr('id', 'coord-' + i + '-' + j);
                // Отрисовка шашек на экране
                if (statesBoard[i][j] === 1)
                {
                    jQuery(this).children().addClass('checker
white_checker');
                }
                else if (statesBoard[i][j] === 2)
                {
                    jQuery(this).children().addClass('checker
black_checker');
                }
                else if (statesBoard[i][j] === 0)
                {
                    jQuery(this).children().removeClass('checker
white_checker black_checker');
                }
                j++;
            });
            i++;
        });
        // Отслеживаем клик по шашке
        self.eventClickChecker();
    };

    /**
     * Событие, отслеживающее клик по шашке
     */
    self.eventClickChecker = function()
    {
        // Сброс предыдущих обработчиков, если они были
        jQuery('.chess').off('click', '.checker');
        if (!self.gameEnd)
        {
```

```
            jQuery('.chess').on('click', '.checker', function()
            {
                var sideChecker = self.getSideChecker();
                if (sideChecker === 1)
                {
                    self.this_checker = self.white_checker;
                }
                else if (sideChecker === 2)
                {
                    self.this_checker = self.black_checker;
                }
                else
                {
                    if (jQuery(this).hasClass(self.white_checker))
                    {
                        self.this_checker = self.white_checker;
                    }
                    else if (jQuery(this).hasClass(self.black_checker))
                    {
                        self.this_checker = self.black_checker;
                    }
                }
                if          (!jQuery(this).hasClass(self.selected)          &&
jQuery(this).hasClass(self.this_checker))
                {
                    jQuery('.checker').removeClass(self.selected);
                    var                 white_checker_class                 =
jQuery(this).hasClass(self.white_checker);
                    var                 black_checker_class                 =
jQuery(this).hasClass(self.black_checker);
                    if (white_checker_class || black_checker_class)
                    {
                        var id = jQuery(this).attr('id'); // Индексы в виде
строки
```

```
var reg = /([0-9]+)/g; // Шаблон индексов
var indexes = convertArrayToInt(id.match(reg)); //
```
Индексы ячейки
```
// Запоминаем координаты выбранной шашки
var i = indexes[0],
    j = indexes[1];
self.setSelectChecker(indexes);
var sideChecker = self.getSideChecker();
if (sideChecker === null)
{
    jQuery(this).addClass(self.selected);
    if (white_checker_class)
    {
        self.setSideChecker(1);
    }
    else if (black_checker_class)
    {
        self.setSideChecker(2);
    }
    self.findRoads(i, j); // Поиск возможных ходов
}
if (sideChecker === 1 && white_checker_class)
{
    jQuery(this).addClass(self.selected);
    self.setSideChecker(1);
    self.findRoads(i, j); // Поиск возможных ходов
}
else if (sideChecker === 2 && black_checker_class)
{
    jQuery(this).addClass(self.selected);
    self.setSideChecker(2);
    self.findRoads(i, j); // Поиск возможных ходов
}
```

```
                    self.showMessage();
                }
            }
        });
    }
};

/**
 * Событие, отслеживающее клик по возможному переходу
 */
self.eventClickRoad = function()
{
    jQuery('.chess').off('click', '.road');
    jQuery('.chess').one('click', '.road', function()
    {
        var statesBoard = self.getStatesBoard();
        // Координаты выбранной шашки
        var selectChecker = self.getSelectChecker();
        var i_cur = selectChecker[0],
            j_cur = selectChecker[1];
        // Новые координаты выбранной шашки
        var id = jQuery(this).attr('id');
        var reg = /([0-9]+)/g;
        var moveChecker = convertArrayToInt(id.match(reg));
        var i_new = moveChecker[0],
            j_new = moveChecker[1];
        // Убираем шашку с текущей позиции
        statesBoard[i_cur][j_cur] = 0;
        // Ставим шашку на новую позицию
        statesBoard[i_new][j_new] = self.getSideChecker();
        // Сбрасываваем возможные ходы для шашек
        jQuery('.checker').removeClass(self.selected);
        self.setStatesBoard(statesBoard);
```

```
        // Удаление шашек соперника
        self.removeChecker();
        // Добавление новых шашек для текущего игрока
        self.addChecker();
        // Сменить текущую сторону
        self.changeSideChecker();
        // Отрисовываем шашки на игровом поле
        self.drawCheckers();
        // Проверка окончания игры
        self.gameOver();
        // Сообщение пользователю
        self.showMessage();
    });
};

/**
 * Отображает сообщение пользователю
 */
self.showMessage = function()
{
    var message = ''; // Сообщение пользователю
    if (self.gameEnd === true)
    {
        message += 'Игра завершена. ';
        if (self.winner === 1)
        {
            message += 'Белые выиграли. ';
        }
        else if (self.winner === 2)
        {
            message += 'Черные выиграли. ';
        }
        else if (self.winner === 0)
```

```
        {
            message += 'Ничья. ';
        }
    }
    else
    {
        if (self.sideChecker === null)
        {
            message += 'Выберите шашку, кликнув по ней ЛКМ. ';
        }
        else if (self.sideChecker === 1)
        {
            message += 'Ходят белые шашки. ';
        }
        else if (self.sideChecker === 2)
        {
            message += 'Ходят черные шашки. ';
        }
        else if (self.sideChecker === 0)
        {
            message += 'Нет ходов. ';
        }
    }
    jQuery('#message').text(message);

};



/**
 * Проверка окончания игры
 * Игра считается завершенной если:
 * 1) Нельзя сходить ни одной шашкой (ничья)
 * 2) Отсутствуют белые шашки (черные выиграли)
```

```
 * 3) Отсутствуют черные шашки (белые выиграли)
 */
self.gameOver = function()
{
    // Количество белых шашек
    var countWhiteCheckers = 0;
    // Количество черных шашек
    var countBlackCheckers = 0;
    // Число возможных переходов для белых шашек
    var roadsWhite = 0;
    // Число возможных переходов для черных шашек
    var roadsBlack = 0;
    // Состояния игровой доски
    var statesBoard = self.getStatesBoard();
    // Проход по всем клеткам игровой доски
    var size = self.getSize();
    for (var i = 0; i < size; i++)
    {
        for (var j = 0; j < size; j++)
        {
            if (statesBoard[i][j] === 1)
            {
                countWhiteCheckers++;
                roadsWhite += self.findRoads(i, j);
            }
            if (statesBoard[i][j] === 2)
            {
                countBlackCheckers++;
                roadsBlack += self.findRoads(i, j);
            }
        }
    }
    self.clearRoad();
```

```
        if (countWhiteCheckers >= 3 && countBlackCheckers <= 2)
        {
            self.winner = 1; // Белые выиграли
        }
        else if (countBlackCheckers >= 3 && countWhiteCheckers <= 2)
        {
            self.winner = 2; // Чёрные выиграли
        }
        else if (countWhiteCheckers <= 2 && countBlackCheckers <= 2 ||
(roadsWhite === 0 && roadsBlack === 0))
        {
            self.winner = 0; // Ничья
        }
        // Проверка окончания игры
        if (self.winner === 1 || self.winner === 2 || self.winner === 0)
        {
            self.gameEnd = true; self.sideChecker = 0;
        }
    };

}

/**
 * Инициализация игровых данных
 */
function init()
{
    var message = ''; // Сообщение пользователю
    var container = 'chess'; // Селектор контейнера для игрового поля
    var sizeBoard = getSizeBoard(); // Размер игрового поля
    var countCheckers = getCountCheckers(); // Количество шашек игрока
    // Проверка входных данных на валидность
    if (sizeBoard === '')
```

```
    {
        message = 'Введите размерность игрового поля.';
    }
    else if (countCheckers === '')
    {
        message = 'Введите число шашек у каждого игрока.';
    }
    else if (!isInt(sizeBoard))
    {
        message = 'Размерность игрового поля дожно быть целым значением.';
    }
    else if (!isInt(countCheckers))
    {
        message = 'Количество шашек игрока дожно быть целым значением.';
    }
    else if (!numPositive(sizeBoard) || sizeBoard < 3)
    {
        message = 'Размерность игрового поля дожно быть положительным значением
и быть не менее 3.';
    }
    else if (!numPositive(countCheckers) || countCheckers < 3)
    {
        message = 'Количество шашек игрока дожно быть положительным значением
и быть не менее 3.';
    }
    else if (!checkMaxSizeBoard(sizeBoard))
    {
        message = 'Введен слишком большой размер игрового поля.';
    }
    else if (!checkMaxCountCheckers(sizeBoard, countCheckers))
    {
        message = 'Введено слишком большое количество шашек для каждого
игрока.';
    }
```

```
    if (message === '')
    { // Данные валидны и ошибок не было
        // Формируем игровое поле
        var board = new Board(container, sizeBoard, countCheckers);
        // Отрисовываем ячейки игрового поля
        board.drawBoard();
        // Отрисовываем шашки на игровом поле
        board.drawCheckers();
    }
    else
    {
        // Очищаем игровое поле
        jQuery('.chess').html('');
        jQuery('#message').html('<br>' + message);
    }
}

/**
 * Выполняем действия когда DOM полностью загружен
 */
jQuery(document).ready(function()
{
    init();
    // Отслеживаем изменение игорового поля и количества шашек
    jQuery('#sizeBoard, #countCheckers').on('change keyup', function()
    {
        init();
    });
});
```

4 Инструкция пользователя

Руководство пользователя – документ, назначение которого предоставить людям помощь в использовании некоторой системы.

Документ входит в состав технической документации на систему и, как правило, подготавливается техническим писателем.

Работа с программным продуктом осуществляется при помощи клавиатуры и мыши.

Шаг 1. Запустить программу

Для этого кликните левой кнопкой мыши по файлу index.html (рисунок 1).

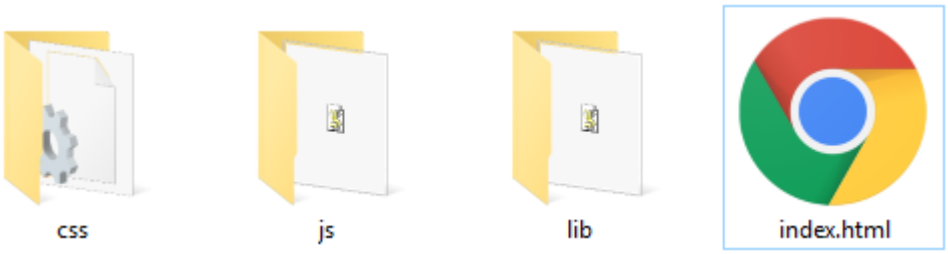

Рисунок 1 – Файл запуска игры

Содержимое файла необходимо открывать в любом современном веб-обозревателе.

Шаг 2. Ознакомиться с правилами игры

Правила игры вы увидите под заголовком «Правила игры» (рисунок 2).

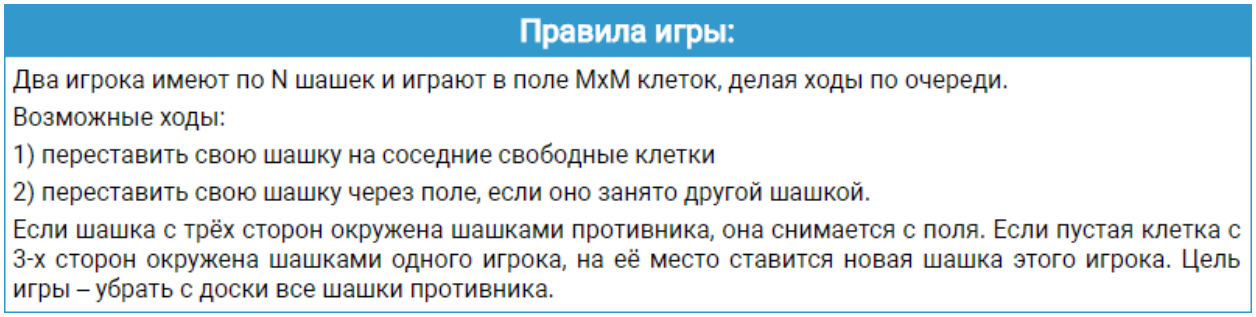

Рисунок 2 – Правила игры

Шаг 3. Выбрать входные параметры

На этом шаге необходимо заполнить два текстовых поля: «Размер игрового поля (M)» и «Число шашек у каждого из игроков (N)» (рисунок 3).

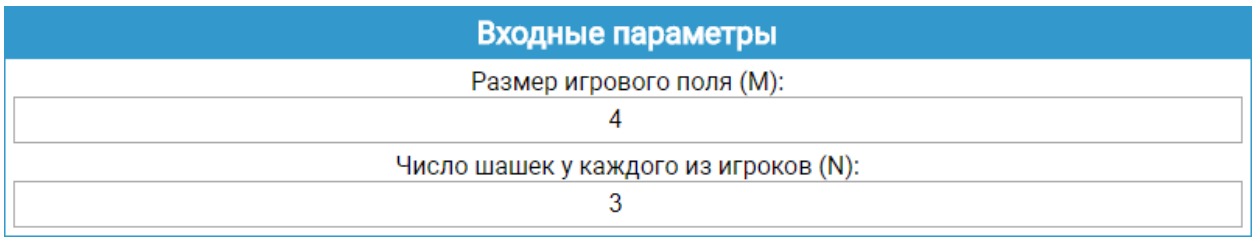

Рисунок 3 – Заполнение входных параметров

Установите фокус на текстовом поле кликнув по нему левой клавишей мыши или при помощи клавиши Tab на клавиатуре. Затем при помощи кнопок справа от значения текстового поля изменить значение, кликнув по ним левой клавишей мыши или покрутив колесико мыши. Эту же операцию можно сделать при помощи клавиатуры двумя способами: использование стрелок на клавиатуре или ввести число при помощи цифровых клавиш.

Допустимый диапазон размера игрового поля от 3 до 9 включительно. Допустимый диапазон числа шашек у каждого из игроков варьируется от размера игровой доски. Если размерность игровой доски четная, то хотя бы один ряд на ней должен быть пустой (без шашек). Если размерность игровой доски нечетная, то пустыми (без шашек) должны быть два ряда. Минимальное количество шашек у каждого из игроков — 3.

Если данные заполнены правильно, то перед вами сформируется игровая доска с шашками (рисунок 4).

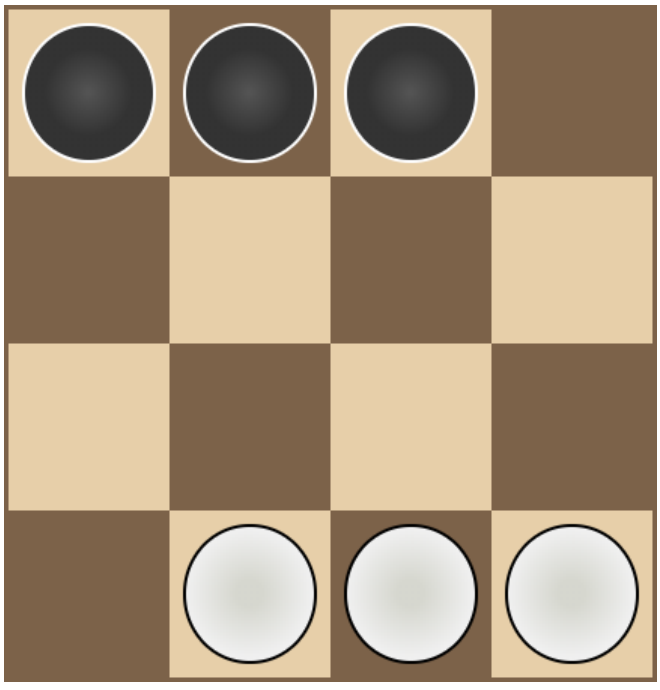

Рисунок 4 – Игровое поле

Шаг 4. Выбрать сторону, выполняющую первый ход

Кликните левой кнопкой мыши по любой из белых или черных шашек. В соответствии с вашим выбором будет определена сторона делающая первый ход.

После первого клика по шашке изменить сторону делающую первый ход нельзя, но можно изменить шашку у выбранной стороны, которая сделает первый ход.

Шаг 5. Выполнять ходы по-очереди пока не будут достигнуты условия победы одной из сторон

Выбор шашки осуществляется при помощи клика по ней левой клавишей мыши. Если шашка выбрана, она обрамляется красным контуром (рисунок 5).

Переход шашки по клеткам осуществляется при помощи клика по области возможного перехода – белый круг внутри клетки (рисунок 5).

Рисунок 4 – Выбранная шашка на игровой доске

После того как одна сторона завершила переход шашкой, ход передается сопернику в том случае если не достигнуты цели игры.

5 Инструкция программиста

Руководство программиста – документ, содержащий информацию для разработчика, которой будет достаточно для создания на базе существующего программного продукта собственных модулей или систем.

Описание функций и методов программного продукта:

isInt()

Функция. Проверка значения на int

numPositive()

Функция. Проверка, является ли число положительным

convertArrayToInt()

Функция. Конвертирует строковые значения массива в целочисленные

getVal()

Функция. Возвращает значение атрибута value по селектору

getSizeBoard()

Функция. Возвращает размер игрового поля

getCountCheckers()

Функция. Возвращает количество шашек для каждого игрока

getAlphabet()

Функция. Возвращает допустимый алфавит для игровой доски

getMaxSizeBoard()

Функция. Возвращает максимальный размер игрового поля

checkMaxSizeBoard()

Функция. Проверяет размер игрового поля с максимально допустимым значением

checkMaxCountCheckers()

Функция. Проверяет количество шашек игрока с максимально допустимым значением

Board()

Класс. Реализует игровое поле

getSize()

Метод класса Board. Возвращает размер игрового поля

getCount()

Метод класса Board. Возвращает количество шашек для одного игрока

setStatesBoard()

Метод класса Board. Запоминает состояния игровой доски

getStatesBoard()

Метод класса Board. Получает состояния игровой доски

setSelectChecker()

Метод класса Board. Запоминает координаты выбранной шашки

getSelectChecker()

Метод класса Board. Получает координаты выбранной шашки

setSideChecker()

Метод класса Board. Запоминает сторону выбранной шашки

getSideChecker()

Метод класса Board. Получает сторону выбранной шашки

getSideCheckerRival()

Метод класса Board. Получает противоположную сторону выбранной шашки

changeSideChecker()

Метод класса Board. Смена хода для другого игрока

createWhiteCheckers()

Метод класса Board. Создает белые шашки в массиве состояний ячеек

createBlackCheckers()

Метод класса Board. Создает черные шашки в массиве состояний ячеек

initStatesBoard()

Метод класса Board. Инициализация состояний ячеек игровой доски

createDivByClass()

Метод класса Board. Формирует тег div с указанным классом

htmlAppend()

Метод класса Board. Добавляет содержимое рядом блочным элементом

htmlReplace()

Метод класса Board. Заменяет содержимое в блочном элементе

htmlReplaceEach()

Метод класса Board. Заменяет содержимое в группе блочных элементов

drawBoard()

Метод класса Board. Отрисовка игрового поля

drawRoad()

Метод класса Board. Отображает возможный переход

findRoads()

Метод класса Board. Поиск возможных ходов для шашки

removeChecker()

Метод класса Board. Удаляет шашку соперника с поля, если она окружена с трех сторон шашками текущего игрока

addChecker()

Метод класса Board. Добавляет шашку на пустую клетку, если она окружена с трех сторон шашками одного игрока

drawCheckers()

Метод класса Board. Отрисовка шашек на игровом поле

eventClickChecker()

Метод класса Board. Событие, отслеживающее клик по шашке

eventClickRoad()

Метод класса Board. Событие, отслеживающее клик по возможному переходу

showMessage()

Метод класса Board. Отображает сообщение пользователю

gameOver()

Метод класса Board. Проверка окончания игры

init()

Функция. Инициализация игровых данных

ready()

Функция. Выполняет действия когда DOM полностью загружен

6 Инструкция администратора

Руководство администратора – документ, содержащий информацию о том, как обеспечить порядок функционирования системы.

Системные требования к программному продукту:

Операционная система

Windows XP/Vista/7/8/10

Mac OS X 10.6 или более поздней версии

Ubuntu 10.04 или более поздней версии

Debian 6 или более поздней версии

OpenSuSE 11.3 или более поздней версии

Процессор

Intel Pentium 4 / Athlon 64 или более поздней версии с поддержкой SSE2

Графический процессор

Intel HD4000 и аналогичные

Жесткий диск

Свободное место на диске 350 Мб или больше

Оперативная память

512 Мб или больше

7 Заключение

В ходе выполнения курсовой работы была разработана программа, реализующая игру «Борьба за жизнь».

Разработан удобный пользовательский интерфейс. Пользователь игровой программы имеет возможность пользоваться как клавиатурой, так и мышью.

Разработаны структуры данных для оперирования состоянием игры, состоянием расположения шашек на игровой доске и смена состояний в соответствии с правилами игры.

К достоинствам системы можно отнести простоту работы с программой и удобную отладку при разработке новых модулей.

Данный проект может дальше развиваться, можно добавлять новые методы, реализующие новые функционал для игры.

Задание, поставленное на курсовую работу, выполнено в полном объеме. Работа полностью соответствует техническому заданию.

8 Используемая литература

1. ГОСТ 34.602-89 Стандарты информационной технологии. Техническое задание на создание автоматизированной системы

2. Илья Кантор. Современный учебник JavaScript, 2015. – 1242 стр. : ил.

3. Макконнелл С. – Совершенный код. Мастер-класс / Пер. с англ. – М. : Издательство "Русская редакция", 2010. – 896 стр. : ил.

4. Методические указания по выполнению лабораторных работ по дисциплине «Технологии программирования» – Нижний Новгород.: НГТУ, 2015. – 25 с.

5. РД 50-34.698-90 Автоматизированные системы. Требования к содержанию документов

6. Стандарт предприятия СТП 1-У-НГТУ-2004. Общие требования к оформлению пояснительных записок дипломных и курсовых проектов. – Нижний Новгород.: НГТУ, 2004. – 22 с.