Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем и программной инженерии

**Лабораторная работа №3**

**по дисциплине**

**«Графический и веб-дизайн»**

**ЯЗЫК HTML. DOM**

**Выполнил**:

ст. гр. ПРИ-120

Д. А. Грачев

**Принял**:

Шамышев А. А.

Владимир, 2023

ЦЕЛЬ РАБОТЫ

Получить навыки работы с языком HTML средствами JavaScript; получить представление о модели DOM.

ВЫПОЛНЕНИЕ РАБОТЫ

1. Был сформирован HTML-файл в котором написана стандартная разметка страницы, стили и установлен обработчик событий на нажатие клавиши

<!doctype *html*>  
<html *lang*="ru">  
<head>  
 <meta *charset*="UTF-8">  
 <meta *name*="viewport"  
 *content*="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">  
 <meta *http-equiv*="X-UA-Compatible" *content*="ie=edge">  
 <title>Лабораторная работа 3</title>  
  
</head>  
<body *onkeydown*="nextIter()" *style*="margin: 0">  
<h1 *class*="count" *style*="position: absolute; top: 0; left: 0; margin: 0; font-size: 42px">0</h1>  
<div *class*="game"></div>  
<script *src*="script.js"></script>  
</body>  
</html>

1. Далее был разработан класс персонажа игры «камень-ножницы-бумага»
   1. Листинг

*class* RPS {  
 *constructor*(rockPaperScissors) {  
 *this*.player = rockPaperScissors  
 *this*.bgColor = *this*.#getColor(rockPaperScissors)  
 *this*.x = Math.round(Math.random() \* 96)  
 *this*.y = Math.round(Math.random() \* 98)  
 }  
  
 #getColor = (rockPaperScissors) => {  
 *switch* (rockPaperScissors) {  
 *case* 'rock':  
 *return* 'black'  
 *case* 'paper':  
 *return* 'green'  
 *case* 'scissors':  
 *return* 'gray'  
 }  
 }  
  
 isBound = (rps) => {  
 *const* xLambda = Math.abs(*this*.x - rps.x)  
 *const* yLambda = Math.abs(*this*.y - rps.y)  
 *return* xLambda <= 2 && yLambda <= 2  
 }  
  
 lose = () => {  
 *if* (*this*.player === 'paper') {  
 *this*.player = 'scissors'  
 *this*.bgColor = 'gray'  
 } *else if* (*this*.player === 'scissors') {  
 *this*.player = 'rock'  
 *this*.bgColor = 'black'  
 } *else* {  
 *this*.player = 'paper'  
 *this*.bgColor = 'green'  
 }  
 }  
  
 move = () => {  
 *const* xRand = Math.random()  
 *const* yRand = Math.random()  
 *if* (xRand > 0.5) *this*.x += *this*.x < 96 ? 1 : -1  
 *else this*.x -= *this*.x > 0 ? 1 : -1  
  
 *if* (yRand > 0.5) *this*.y += *this*.y < 98 ? 1 : -1  
 *else this*.y -= *this*.y > 0 ? 1 : -1  
 }  
}

* 1. Описание работы: для инициализации объекта в его конструктор передается строкой тип персонажа(камень, ножницы или бумага), после чего ему устанавливается цвет(камень – черный, ножницы – серые, бумага - зеленая).  
     У класса есть несколько методов: isBound отвечает за проверку пересечения с другим персонажами для определения сражения; lose – меняет тип персонажа после поражения; move – передвигает персонажа в рандомное место на единицу, так же не позваляя уйти за края

1. Созданы вспомогательные функции для работы игры
   1. getPlayer()
      1. Листинг

*function* getPlayer() {  
 *const* rand = Math.random()  
 *if* (rand > 0.666) {  
 *return* 'rock'  
 }  
 *if* (rand < 0.333) {  
 *return* 'paper'  
 }  
 *return* 'scissors'  
}

* + 1. Описание работы: используется при рандомизации создания персонажа и на основе рандома вибирает тип персонажа
  1. whoWin(p1, p2)
     1. Листинг

*function* whoWin(p1, p2) {  
 *if* (p1.player === 'paper' && p2.player === 'rock') {  
 *return* 1  
 }  
 *if* (p1.player === 'paper' && p2.player === 'scissors') {  
 *return* 2  
 }  
 *if* (p1.player === 'paper' && p2.player === 'paper') {  
 *return* -1  
 }  
  
 *if* (p1.player === 'rock' && p2.player === 'paper') {  
 *return* 2  
 }  
 *if* (p1.player === 'rock' && p2.player === 'scissors') {  
 *return* 1  
 }  
 *if* (p1.player === 'rock' && p2.player === 'rock') {  
 *return* -1  
 }  
  
 *if* (p1.player === 'scissors' && p2.player === 'paper') {  
 *return* 1  
 }  
 *if* (p1.player === 'scissors' && p2.player === 'rock') {  
 *return* 2  
 }  
 *if* (p1.player === 'scissors' && p2.player === 'scissors') {  
 *return* -1  
 }  
}

* + 1. Описание работы: во время сражения высчитывает кто победил на основе обычных правил «камень-ножницы-бумага»
  1. delay(ms)
     1. Листинг

*const* delay = *async* (ms) => *await new* Promise(resolve => setTimeout(resolve, ms));

* + 1. Описание работы: передается количество миллисекунд для создания задержки в работе программы
  1. render(players)
     1. Листинг

*function* render(players) {  
 *const* game = document.body.querySelector('.game')  
 game.innerHTML = ''  
 players.forEach((rps) => {  
 *const* div = document.createElement('div')  
 div.style.width = '30px'  
 div.style.height = '30px'  
 div.style.backgroundColor = rps.bgColor  
 div.style.position = 'absolute'  
 div.style.top = `${rps.y}%`  
 div.style.left = `${rps.x}%`  
 div.style.borderRadius = '30%'  
 div.dataset.player = rps.player  
 game.append(div)  
 })  
}

* + 1. Оисание работы: принимает в себя массив персонажей после чего отображает их на экране с установленными стилями
  1. nextIter()
     1. Листинг

*function* nextIter() {  
 *for* (*let* i = 0; i < players.length - 2; i++) {  
 *for* (*let* j = i + 1; j < players.length - 1; j++) {  
 *const* p1 = players[i]  
 *const* p2 = players[j]  
  
 *if* (p1.isBound(p2)) {  
 *const* winner = whoWin(p1,p2)  
 *if* (winner === 1) {  
 p2.lose()  
 renderCount()  
 }  
 *if* (winner === 2) {  
 p1.lose()  
 renderCount()  
 }  
 }  
 players[i] = p1  
 players[j] = p2  
 }  
 }  
  
 players.forEach((rps) => rps.move())  
 render(players)  
}

* + 1. Описание работы: функция, которая отвечает за игру с клавиатуры, срабатывает при нажатии клавиши, так же высчитывает сражения и победивших

1. Код выполнения начала скрипта:
   1. Листинг

*let* count = 0  
*let* players = [*new* RPS('paper')]  
*for* (*let* i = 0; i < 20; i++) {  
 players.push(*new* RPS(getPlayer()))  
}  
render(players)  
delay(2000).then(() => {  
 *const* playersCopy = [...players]  
 playersCopy.forEach((rps) => rps.move())  
 players = [...playersCopy]  
 render(players)  
})

* 1. Описание работы: создается переменная для счета количества сражений, массив персонажей, который далее заполняется, далее происходит рендер и перерендер через 2 секунды

1. Результат работы программы представлен на скриншоте ниже в состоянии середины игры, на котором видно количество сражений слева вверху

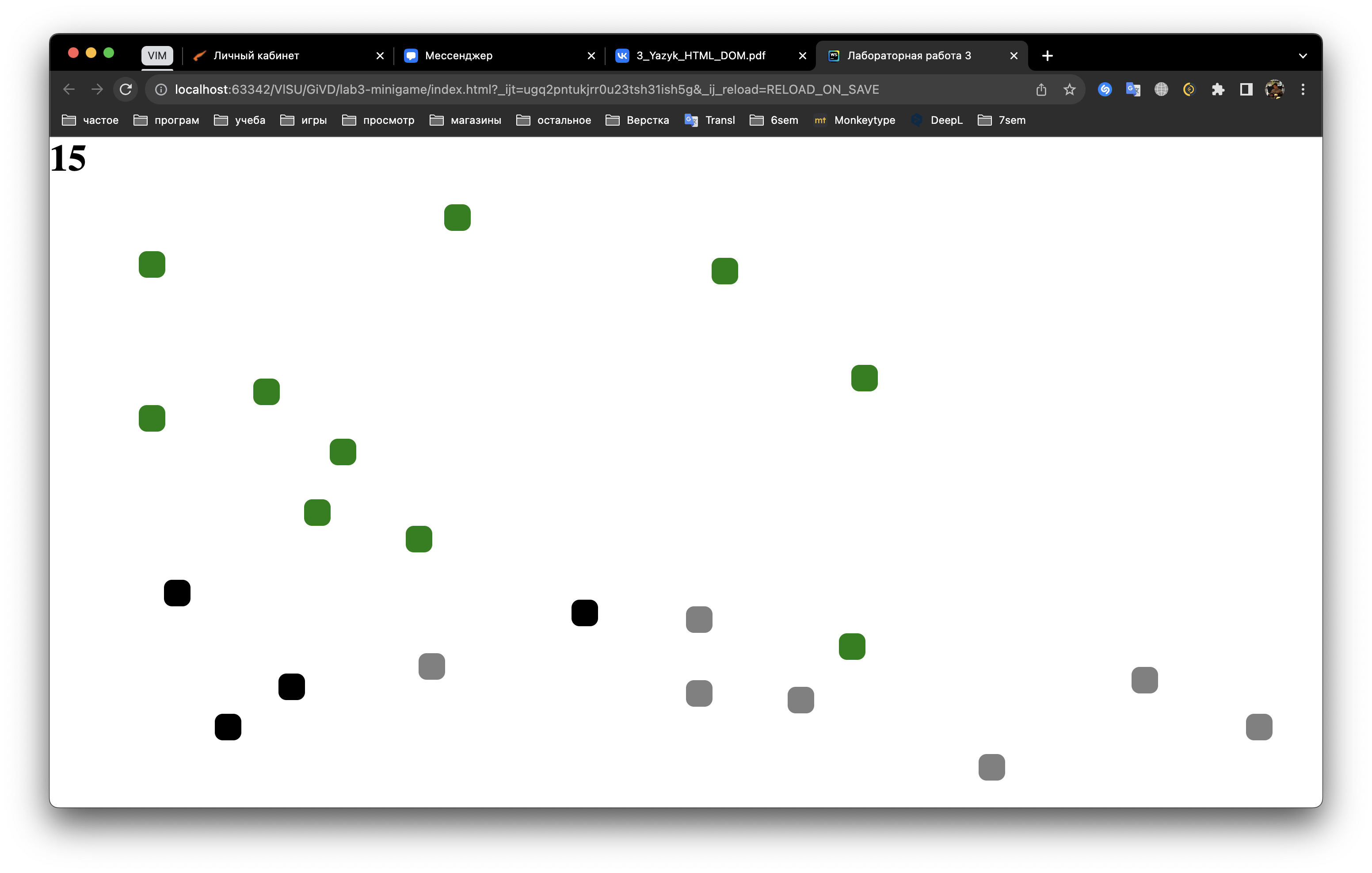


Рисунок . Результат работы

ВЫВОД

В ходе выполнения работы были получены навыки работы с языком HTML средствами JavaScript, получено представление о модели DOM