**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ГОРОДА МОСКВЫ ДОПОЛНИТЕЛЬНОГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ ЦЕНТР ПРОФЕССИОНАЛЬНЫХ КВАЛИФИКАЦИЙ И СОДЕЙСТВИЯ ТРУДОУСТРОЙСТВУ «ПРОФЕССИОНАЛ»**

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к итоговой аттестационной работе на тему

**«Разработка web-игры с использованием технологий HTML, CSS, JavaScript»**

(https://github.com/IgorMonakhov/Attestation

http://shoemaker.p-host.in/)

слушателя Монахова Игоря Германовича группы №: 0207

программы профессиональной переподготовки

«Frontend разработка»

Москва, 2022

**Оглавление.**

# 1.Постановка задачи и план работы.

Данная Итоговая аттестационная работа направлена на практическую реализацию знаний, полученных в результате прослушивания учебной программы профессиональной переподготовки «Frontend разработка».

Основной задачей аттестационной работы было создать игру, способную корректно работать в наиболее распространенном веб-браузере на стационарном персональном компьютере.

Этапами выполнения данной работы было следующее:

- определение общей концепции игры;
- определение размещения и движение игровых объектов на экране;
- определение результатов событий взаимодействия объектов;
- написание кода продукта в VSCode с применением языка JavaScript и его отладка в веб-браузере - Google Chrome;
- оформление интерфейса средствами языка стилей CSS и обеспечение адаптивности к различным размерам дисплеев.

# 2.Основная часть.

Назначением представленного веб-ресурса является страница с игрой, размещенной на хостинге. Целью игры является управление автомобилем и избегание столкновений со случайно появляющимися препятствиями. Управление автомобилем осуществляется пользователем при помощи перемещения компьютерной мыши. Любое столкновение с препятствием завершает игру. На экране игрок видит время, которое удалось продержаться в игре. Игрок может повторить попытку, нажав клавишу F5.

Разработка данного приложения состояла из нескольких этапов.

На первом этапе создания продукта была продумана общая концепция игры (Рис.1), а именно - движение автомобиля по дороге со случайно возникающими препятствиями. Была предусмотрена возможность пользовательского управления, а также было определено сколько и какие объекты будут представлены на экране компьютера пользователя. В центре игрового окна будет расположен автомобиль, способный откликаться на перемещение мыши.

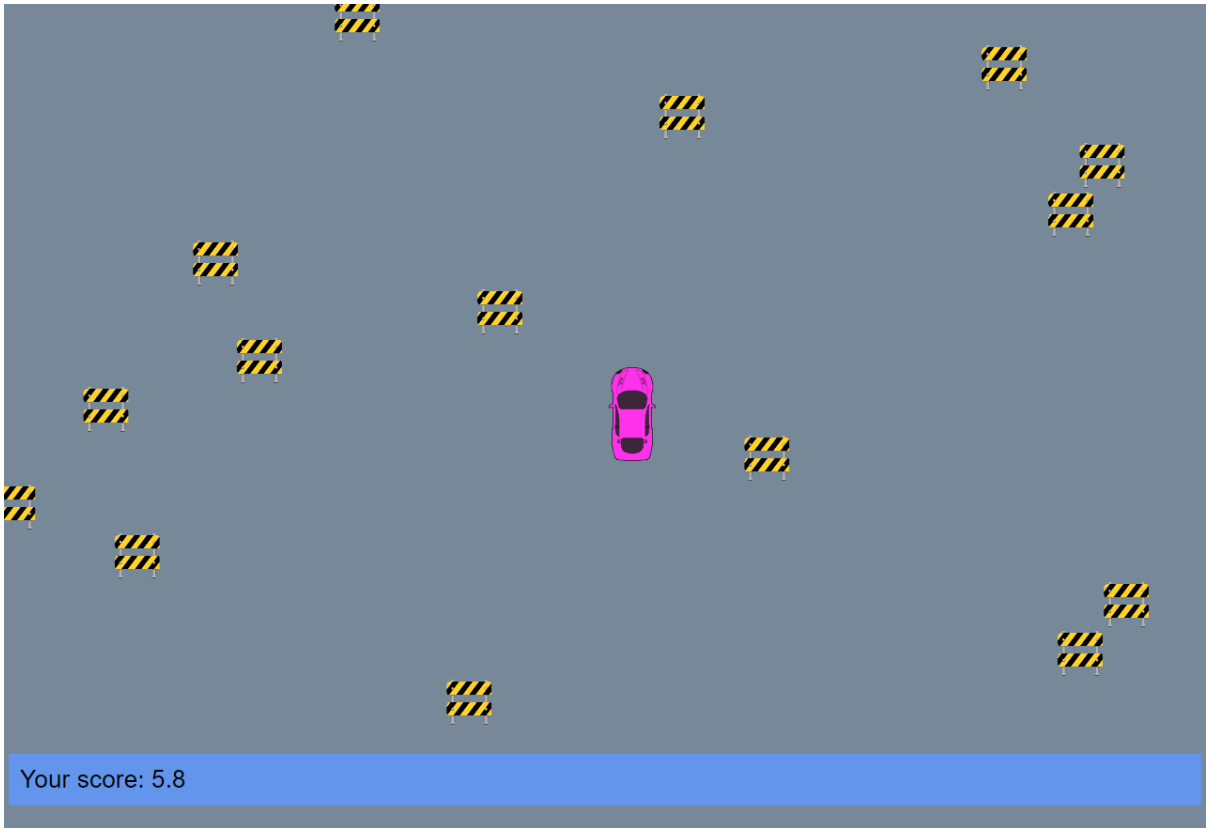


Рис.1

Вторым этапом разработки было определить порядок взаимодействия с автомобилем набегающих на него объектов, а именно реализовать столкновение с препятствиями с остановкой игры.

На третьем этапе разработки для визуализации игры был выбран наиболее распространенный веб-браузер - Google Chrome, а в качестве языка программирования - JavaScript. Был разработан и протестирован программный продукт.

Основой игры послужила HTML разметка файла index.html (Рис.2)

```
1   <!DOCTYPE html>
2   <html lang="en">
3
4   <head>
5       <meta charset="UTF-8">
6       <meta http-equiv="X-UA-Compatible" content="IE=edge">
7       <meta name="viewport" content="width=device-width, initial-scale=1.0">
8       <title>Attestation work</title>
9       <link rel="stylesheet" type="text/css" href="css/style.css"> <!--
        Подключаем внешнюю таблицу стилей -->
10  </head>
11
12  <body>
13
14      <canvas id="scene_canvas"></canvas>
15      <div class="footer">
16          <div class="box-score">Your score:
17              <span id="score">0</span>
18          </div>
19          <div class="box-endofgame">
20              <span id="endofgame">GAME OVER! Press F5 to restart</span>
21          </div>
22      </div>
23
24  </body>
25  <script src="js/script.js"></script> <!-- Подключаем сценарии -->
26
27  </html>
```

Рис.2

В программном коде был применен элемент HTML5 «Canvas» (Строка 14. Рис.2), предназначенный для создания растрового двухмерного изображения при помощи скриптов, обычно на языке JavaScript

К файлу «index.html» был подключен файл «script.js», содержащий код на языке JavaScript с подробными комментариями, реализующий создание, движение и взаимодействие объектов на холсте для рисования «Canvas» (Рис.3).

```
    // Холст для рисования изображений
    let canvas = document.getElementById("scene_canvas")
```

```
// Текущее количество очков
let score = 0

// HTML-элемент для отображения количества очков
let score_element = document.getElementById("score")
let endofgame_element = document.getElementById("endofgame")

// Число, задающее частоту кадров.
// Означает количество времени в секундах, которое занимает обработка 1 кадра
// При значении 0.01 частота кадров будет 100 FPS (в идельных условиях)
let frame_time = 0.01

// Ширина дороги
let bodySize = document.body.getBoundingClientRect();
canvas.width = bodySize.width * 1;
canvas.height = bodySize.height * 0.9;

// Ширина холста в пикселях
function get_width() {
    return canvas.clientWidth
}

// Высота холста в пикселях
function get_height() {
    return canvas.clientHeight
}

// Массив X-координат препятствий
let obstacles_x = []

// Массив Y-координат препятствий
let obstacles_y = []

// Вертикальная скорость движения препятствия (пикселей в секунду)
let obstacle_speed = 200

// Координаты автомобиля. Изначально располагается в центре холста
let car_x = get_width() / 2
let car_y = get_height() / 2

// Ширина и высота автомибя в пикселях
let car_width = 50
let car_height = 100

// Изображение автомобиля
let car_image = new Image()
car_image.src = "images/Car.png"

// Ширина и высота препятствия в пикселях
let obstacle_width = 50
let obstacle_height = 50

// Изображение препятствия
let obstacle_image = new Image()
obstacle_image.src = "images/Barricade.png"
```

```
// Время в секундах до появления следующего препятствия. Обновляется каждый кадр
let time_to_next_obstacle = 0

// Время в секундах между появлением препятствий. Не изменяется.
let obstacle_spawn_period = 0.25

// Объект для отображения трёхмерной графики
let ctx = canvas.getContext("2d")

// При движении мыши получаем координату курсора и записываем в координаты автомобиля
canvas.onmousemove = (event) => {
    event = event || window.event; // IE fix

    car_x = event.pageX
    car_y = event.pageY
}

// Функция отрисовки изображения
// image - изображение
// x_location - X координата на холсте в пикселях
// y_location - Y координата на холсте в пикселях
// width - ширина изображения в пикселях
// height - высота изображения в пикселях
function draw_image(image, x_location, y_location, width, height) {

    // Координаты изображения нужно сместить на половину ширины и высоты,
    // т.к. браузер рисует картинку начиная с её левого верхнего угла
    let x = x_location - width / 2
    let y = y_location - height / 2

    // Библиотечная функция рисования
    ctx.drawImage(image, x, y, width, height)
}

// Вспомогательная функция для рисования автомобиля
function draw_car() {
    draw_image(car_image, car_x, car_y, car_width, car_height)
}

// Вспомогательная функция для рисования препятствия по указанному номеру
function draw_obstacle(index) {
    // Получаем координаты препятствия по номеру из массивов
    let x = obstacles_x[index]
    let y = obstacles_y[index]

    draw_image(obstacle_image, x, y, obstacle_width, obstacle_height)
}

// Вспомогательная функция для получения случайного числа в диапазоне [min, max]
function random_range(min, max) {
    return Math.random() * (max - min) + min
}

// Создать новое препятствие на верху холста со случайной X координатой
function create_obstacle() {
    let x_min = 0
    let x_max = get_width()
```

```
        let y = 0

        // Получаем случайную X координату между 0 и шириной холста
        let x = random_range(x_min, x_max)

        // Добавляем новые координаты препятствия в массивы
        obstacles_x.push(x)
        obstacles_y.push(y)
    }

    // Удаляет препятствие по номеру
    function remove_obstacle(index) {
        obstacles_x.splice(index, 1)
        obstacles_y.splice(index, 1)
    }

    // Рисование всех препятствий, смещение их вниз по холсту и определение столкновения с
игроком
    function update_obstacles() {
        // Идём по массиву всех препятствий.
        // Используем массив obstacles_x, хотя можно и obstacles_y, их длина одинаковая
        for (let i = 0; i < obstacles_x.length; i++) {
            // Рисуем препятствие с номером i
            draw_obstacle(i)

            // Смещаем i-ое препятствие на obstacle_speed * frame_time вниз
            // obstacle_speed - это пиксели в секунду
            // Чтобы получить скорость в пикселях в КАДР, нужно умножить это значение
            // на время одного кадра.
            obstacles_y[i] += obstacle_speed * frame_time

            // Определяем расстояние от i-ого препятствия до игрока
            let dx = car_x - obstacles_x[i]
            let dy = car_y - obstacles_y[i]

            let distance_to_car = Math.sqrt(dx * dx + dy * dy)

            // Если от i-го препятствия до игрока менее 50 пикселей, то игра закончена
            if (distance_to_car < 50) {
                // TODO Collision!
                console.log("Collision!")
                game_over()
            }

            // Если препятствие вылезло за пределы холста, удаляем его
            if (obstacles_y[i] >= get_height()) {
                remove_obstacle(i)

                // Т.к. размер массива изменился после удаления i-го препятствия,
                // нужно уменьшить i на единицу, чтобы не пропустить следующее препятствие
                i--
            }
        }
    }

    // Функция обработки кадра
    // Вызывается 100 раз в секунду, в теории достигая частоты кадров в 100 FPS
```

```
    function update_loop() {
        // Очищаем холст, удаляя всё, что было нарисовано в предыдущем кадре
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Изменяем время до появления следующего препятствия
        if (time_to_next_obstacle <= 0) {
            // Если таймер дошёл до нуля, создаем новое препятствие
            create_obstacle()

            // И перезапускаем таймер, записывая в него период появления препятствий
            time_to_next_obstacle = obstacle_spawn_period
        }
        else {
            // Если таймер еще не дошел до нуля, вычитаем из него время кадра.
            // Таким образом счётчик таймера будет уменьшатся на 0,01 каждые 10
миллисекунд,
            // что равно 1 каждую секунду.
            // Таким способом можно измерять прошедшее реальное время
            time_to_next_obstacle -= frame_time
        }

        // Рисуем препятствия, смещаем их вниз и определеяем столкновение с игроком
        update_obstacles()

        // Рисуем автомобиль
        draw_car()

        // Увеличиваем счётчик очков на время кадра
        // Таким образом количество очков будет равно времени игры в секундах
        score += frame_time

        // Выводим количество очков в HTML-элемент.
        // .toFixed(1) ограничивает количество символов после точки до 1
        // Например, число 3.1215 будет выведено как 3.1
        score_element.innerText = score.toFixed(1)
    }

    // Вызывается при поражении
    function game_over() {
        // Останавливаем обработку кадров. Игра полностью останавливается, перестают
двигаться
        // препятствия, игрок, не определяются столкновения и т.д.
        clearInterval(timer_handler)
        endofgame_element.style.visibility = "visible"
    }

    // Устанавливаем интервальный таймер для функции update_loop()
    // Браузер будет вызывать функцию update_loop каждые frame_time секунд
    // setInterval принимает время в миллисекундах, поэтому умножаем frame_time на 1000
    let timer_handler = setInterval(() => update_loop(), frame_time * 1000)
```

Рис.3

Четвертым этапом проведено оформление интерфейса с применением языка стилей CSS (Рис.4).

```css
html,
body {
  margin: 0;
  padding: 0;
  overflow: hidden;
}

body {
  width: 100%;
  height: 100vh;
  background-color: lightslategray;
}

.footer {
  margin: 5px;
  background-color: cornflowerblue;
  padding: 12px;
  font: 1.5em sans-serif;
  display: block;
}

.box-score,
.box-endofgame {
  display: inline-block;
  width: 33.333%;
  font-size: 3vmin;
}

.box-endofgame {
  text-align: center;
  visibility: hidden;
  color: rec;
}
```

Рис.4

На пятом этапе все файлы с исходным программным кодом и файлы с картинками были размещены в собственном репозитории GitHub: https://github.com/IgorMonakhov/Attestation

При этом, для целей демонстрации работоспособности программы все файлы приложения были также размещены на хостинге: http://shoemaker.p-host.in/

На шестом, заключительном этапе было осуществлено тестирование программного продукта на дисплеях разных разрешений (Рис.5).


Рис.5

# 3.Список литературы


1) Современный учебник JavaScript. https://learn.javascript.ru/
2) Изучайте CSS. Постоянно обновляемый курс и справочник по CSS для повышения вашего уровня знаний в сфере веб-дизайна. https://web.dev/learn/css/
3) Resources for Developers, by Developers. Documenting web technologies, including CSS, HTML, and JavaScript, since 2005. https://developer.mozilla.org/ru/
4) Самоучитель HTML5. http://htmlbook.ru/samhtml5