Федеральное государственное бюджетное образовательное учреждение высшего образования «Чувашский государственный университет им. И.Н. Ульянова»

Кафедра вычислительной техники

Лабораторная работа №5

по дисциплине WEB-программирование

ДИНАМИЧЕСКИЕ WEB-СТРАНИЦЫ. ОБРАБОТКА СОБЫТИЙ.

Выполнил: Кузнецов Даниил Андреевич  
студент группы ИВТ-42-23

Проверила: Первова Н.В.

Чебоксары, 2025

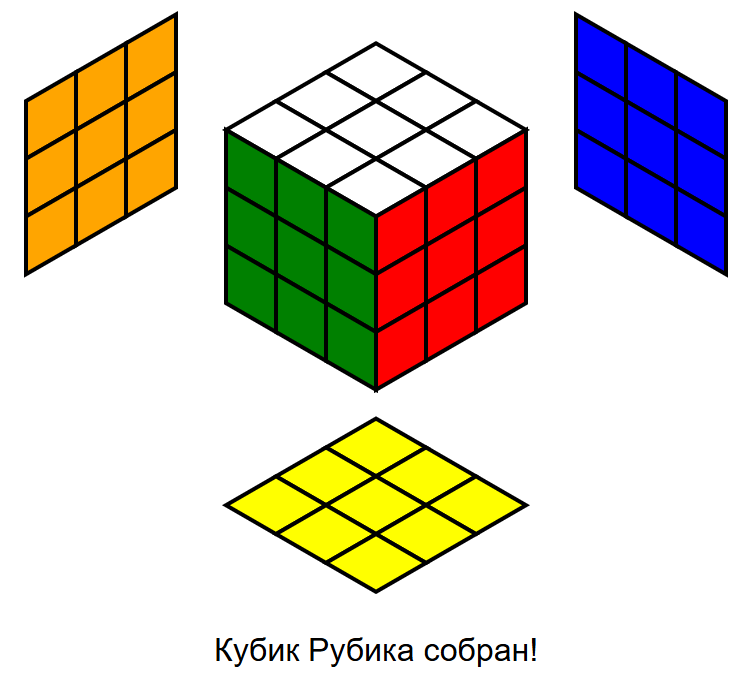
**Цель работы**

Изучить принципы разработки динамических WEB-страниц с использованием клиентских сценариев JavaScript и объектной модели документа. Научиться создавать кроссбраузерные HTML-страницы. Познакомиться с уровнями и особенностями DOM API и обработкой событий DOM различных уровней.

**Задание**

Разработать игровую веб-страницу на тему: кубик Рубика.

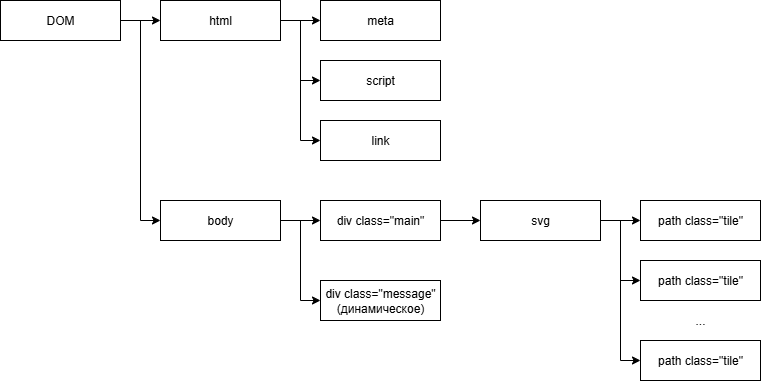
**Сценарий разрабатываемой игры и её интерфейс**



На экране расположена развёртка Кубика Рубика. По нажатии клавиш R, M, L, U, E, D, F, S, B происходит вращение соответствующей грани: правой, средней (передней верхней), левой, верхней, средней (передней справа), нижней, передней (верхней справа), средней, задней. При нажатии клавиши Backspace или изменении рзамеров окна кубик замешивается. Также вращения возможны при помощи свайпов (мышью или пальцем). В собранном состоянии выводится сообщение: «Кубик Рубика собран!». Интерфейс игры представлен на рисунке.

**Описание интерфейса в терминах DOM-модели**

|  |  |  |  |
| --- | --- | --- | --- |
| **Элемент интерфейса** | **HTML-элемент** | **ID/Класс** | **Описание** |
| Окно игры | div | id = “main” | Пространство для основного интерфейса |
| Картинка с кубиком | svg | – | На картинке с кубиком присутствуют все грани, необходимые для сборки кубика |
| Стикер | path | class = “tile” | Стикер имеет цвет и положение на экране |
| Сообщение | div | class = “message” | Сообщение о победе, которое появляется динамически («Кубик Рубика собран!») |



**Краткое описание игрового сценария**

**Основные классы:**

RubiksCube - основной класс, представляющий состояние кубика

**Константы для вычислений поворотов:**

TURN\_DEGREE\_COMPUTE - преобразование углов в команды поворота граней  
ROTATION\_DEGREE\_COMPUTE - преобразование углов в команды вращения кубика  
TURN\_COMPUTE - матрица преобразования жестов в команды для каждой грани  
ROTATION\_COMPUTE - преобразование углов в команды вращения

**Переменные состояния:**

rubiksCube - экземпляр класса кубика Рубика  
mouseData - отслеживание состояния взаимодействия  
click - координаты начальной точки жеста  
sticker - информация о выбранном стикере (грань, координаты)

**Обработка событий:**

1. События мыши/тачскрина

Последовательность обработки:

mousedown/touchstart:

1. Запись начальных координат касания
2. Определение выбранного стикера (если касание на грани кубика)

mouseup/touchend:

1. Вычисление расстояния и угла жеста
2. Определение типа операции:
3. Поворот грани (если выбран стикер)
4. Вращение кубика (если не выбран стикер)
5. Выполнение соответствующего хода через rubiksCube.move()
6. Обновление отображения через updateStickers()

2. События клавиатуры

keydown:

1. Backspace - перемешивание кубика (rubiksCube.scramble())
2. Буквенные клавиши (R, M, L, U, E, D, F, S, B) - выполнение ходов
3. Shift + буква - обратные ходы
4. Обновление отображения после каждого хода

3. Другие события

resize - автоматическое перемешивание при изменении размера окна

**Алгоритм обработки жестов:**

1. Вычисление угла между начальной и конечной точками жеста
2. Классификация жеста:
3. Определение квадранта угла через getTurnAngle()/getRotationAngle()
4. Преобразование в команду через таблицы TURN\_COMPUTE/ROTATION\_COMPUTE
5. Выполнение команды через систему движений кубика

**Проверка состояния:**

updateStickers():

1. Обновление цветов всех стикеров
2. Проверка собранности кубика через rubiksCube.isSolved()
3. Отображение сообщения о победе при успешной сборке

document

Описание: каждая веб-страница, которая загружается в браузер, имеет свой собственный объект document. Интерфейс документа служит точкой входа для получения содержимого веб-страницы (всего DOM - дерева, включая такие элементы как <body> и <table>), а также обеспечивает функциональность, которая является глобальной для документа, например, для получения URL-адреса страницы или создания новых элементов в документе).

Доступ к узлам DOM-дерева:

document.getElementById()

Описание: Возвращает ссылку на элемент по его идентификатору (ID); идентификатор является строкой, которая может быть использована для идентификации элемента; она может быть определена при помощи атрибута id в HTML или из скрипта.

Синтаксис: element = document.getElementById(id);

Параметры:

• elementId (String) - уникальный идентификатор элемента

Возвращаемое значение: Element или null

Пример использования:

tbodyEl = document.getElementById('board') для получения ссылки на тело таблицы.

Особенности:

• Поиск выполняется только среди элементов, уже добавленных в документ

• ID должны быть уникальными в пределах документа

• чувствителен к регистру

document.getElementsByTagName()

Описание: возвращает HTMLCollection элементов с указанным именем тега. Поиск осуществляется по всему документу, включая корневой узел. Возвращаемая HTMLCollection живая, это значит что она автоматически обновляет сама себя чтобы оставаться синхронизированной с DOM деревом без необходимости вызова document.getElementByTagName() снова.

Синтаксис: var elements = document.getElementsByTagName(name);

Работа с параметрами HTML-элементов и их CSS-свойствами:

element.textContent

Описание: свойство для получения или установки текстового содержимого узла и его потомков.

Синтаксис: var text = element.textContent;

element.textContent = "Это просто текст";

Пример использования:

movesEl.textContent = String(moves) для задания текущего колва ходов

title.textContent = 'Ты собрл!' для получения текста при прохождении головоломки

element.style

Описание: Свойство style только для чтения в HTMLElement возвращает встроенный стиль элемента в виде динамического CSSStyleDeclaration-объекта, который содержит список всех свойств стиля для этого элемента со значениями, присвоенными только тем атрибутам, которые определены во встроенном style-атрибуте элемента.

Пример использования:

btn.style.backgroundImage = "url(image/"+val+".png)"

btn.style.backgroundSize = 'cover'

btn.style.backgroundPosition = 'center'

element.setAttribute()

Описание: Метод setAttribute() интерфейса Element устанавливает значение атрибута для указанного элемента. Если атрибут уже существует, то его значение будет обновлено, а если нет, то будет добавлен атрибут с соответствующим именем и значением.

Синтаксис: element.setAttribute(name, value)

Параметры:

• name (String) - имя атрибута

• value (String) - значение атрибута

Пример использования:

btn.setAttribute('data-index', String(idx))

btn.setAttribute('type', 'button')

btn.setAttribute('aria-label', val === 0 ? 'Пустая клетка' : 'Плитка ' + val)

element.classList

Описание: Element.classList — это доступное только для чтения свойство, которое содержит текущую коллекцию DOMTokenList всех атрибутов class элемента. Использование classList представляет более удобный способ, чем доступ к списку классов элемента в виде строки, разделенной пробелами, через element.className.

Методы:

• add() - добавление класса

• remove() - удаление класса

• contains() - проверка наличия класса

Пример использования:

btn.classList.add('empty')

Модификация DOM-дерева:

document.createElement()

Описание: Создает новый HTML-элемент с указанным именем тега.

Синтаксис: var element = document.createElement(tagName, [options]);

element — созданный объект элемента.

tagName — строка, указывающая элемент какого типа должен быть создан. nodeName создаётся и инициализируется со значением tagName.

options — необязательный параметр, объект ElementCreationOptions, который может содержать только поле is, указывающее имя пользовательского элемента, созданного с помощью customElements.define()

Возвращаемое значение: новый Element

Пример использования:

const tr = document.createElement('tr')

const btn = document.createElement('button')

parentNode.appendChild()

Описание: Node.appendChild() добавляет узел в конец списка дочерних элементов указанного родительского узла. Если данный дочерний элемент является ссылкой на существующий узел в документе, то функция appendChild() перемещает его из текущей позиции в новую позицию (нет необходимости удалять узел из родительского узла перед добавлением его к какому-либо другому узлу).

Синтаксис: var child = element.appendChild(child);

Возвращаемое значение: Возвращается дочерний элемент (aChild), кроме тех случаев, когда child это DocumentFragment, в таком случае возвращается пустой DocumentFragment.

Пример использования:

td.appendChild(btn)

tr.appendChild(td)

EventTarget: метод addEventListener()

Описание: Метод addEventListener() интерфейса EventTarget настраивает функцию, которая будет вызываться всякий раз, когда указанное событие будет доставлено целевому объекту.

Синтаксис: addEventListener(type, listener)

addEventListener(type, listener, options)

addEventListener(type, listener, useCapture)

Параметры:\

Type: Строка с учётом регистра, обозначающая тип события, которое нужно отслеживать.

Listener: Объект, получающий уведомление (объект, реализующий интерфейс Event), когда происходит событие указанного типа. Это должен быть null, объект с методом handleEvent() или JavaScript функция.

Пример использования:

btn.addEventListener('mousemove', positionTooltip)

btn.removeEventListener('mousemove', positionTooltip)