# **Найди пару**

# Цель урока

Сделать игру “Найди пару”, попрактиковаться в текущих знаниях и приобрести новые.

# Содержание

1. [CSS](#_3dy6vkm)
2. [Анимаци](#_1t3h5sf)я
3. [Переворачиваем карты](#_4d34og8)
4. [Сравнение карт](#_17dp8vu)
5. [Блокировка карт](#_3rdcrjn)
6. [Великий рандом](#_26in1rg)
7. [Домашнее задание](#_1ksv4uv)

# Что повторяем

Стандартные css свойства, метод массива forEach, условие, тернарный оператор

# Ссылки на материалы и личный кабинет

Рабочие материалы

Материалы

Урок в личном кабинете

# Задачи

* Повторить пройденный материал
* Научиться работать с анимацией в css
* узнать новые свойства в css
* Познакомиться с addEventListener, removeEventListener

# Ход урока

Подготовим HTML. Для начала как обычно подготавливаем шаблон, подключаем к нему css и js.

|  |
| --- |
| **<!DOCTYPE html> <html lang="en"> <head>  <meta charset="UTF-8">   <title>Find pair</title>   <link rel="stylesheet" href="style.css"> </head> <body>     <script src="script.js"></script> </body> </html>** |

CSS будем делать на флексах, поэтому нужен элемент, который будет выполнять роль контейнера. Мы возьмем section. Объясните ребенку, что это тоже самое, что div. Отличие у них чисто семантическое. Section применяется для всякий диалоговых окон. Тег новый, поэтому его стоит использовать аккуратно, есть браузеры, которые его не понимают, а значит все стили к нему они тоже проигнорируют.

У нас 12 карт, поэтому мы сделаем 12 тегов див. Для контейнера дадим class=’game’ , а для его элементов ‘card’

|  |
| --- |
| **<!DOCTYPE html>** <html lang="en"> <head>  <meta charset="UTF-8">   <title>Find pair</title>   <link rel="stylesheet" href="style.css"> </head> <body>  <section class="game">  <div class="card" >    </div>  <div class="card" >    </div>   <div class="card" >    </div>  <div class="card" >    </div>   <div class="card" >    </div>  <div class="card" >    </div>   <div class="card" >    </div>  <div class="card" >    </div>   <div class="card" >    </div>  <div class="card" >    </div>   <div class="card" >    </div>  <div class="card" >    </div>  </section>   <script src="script.js"></script> </body> </html> |

Отлично, теперь порассуждаем. У нас должна быть картинка рубашки карты, а потом карта переворачивается и показывается другая картинка. Значит сколько нам картинок нужно в каждом диве? Правильно, в каждом по две. В каждый див добавляем одну картинки лицевой стороны карты и одну - рубашки. Предлагаю картинке с лицевой стороной дать класс front-face и для рубашки back-face.

|  |
| --- |
| **<!DOCTYPE html>** <html lang="en"> <head>  <meta charset="UTF-8">   <title>Find pair</title>   <link rel="stylesheet" href="style.css"> </head> <body>  <section class="game">  <div class="card" >  <img class="front-face" src="img/1.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" >  <img class="front-face" src="img/1.svg" />  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" >  <img class="front-face" src="img/2.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" >  <img class="front-face" src="img/2.svg" />  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" >  <img class="front-face" src="img/3.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" >  <img class="front-face" src="img/3.svg" />  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" >  <img class="front-face" src="img/4.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" >  <img class="front-face" src="img/4.svg" />  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" >  <img class="front-face" src="img/5.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" >  <img class="front-face" src="img/5.svg" />  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" >  <img class="front-face" src="img/6.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" >  <img class="front-face" src="img/6.svg" />  <img class="back-face" src="img/back.svg" />  </div>  </section>   <script src="script.js"></script> </body> </html> |

Пока что HTML у нас готов. Нужно сделать из этого то, что у нас на макете (он в папке img)



## CSS

Для начала всегда полезно сделать сброс margin и padding.

|  |
| --- |
| \* {  padding: 0;  margin: 0;  box-sizing: border-box; } |

Селектор \* значит, что мы обращаемся к document.

Если ученик забыл что такое box-sizing: border-box,

то объясняем, что это просто когда размер padding и boeder входит в высоту и ширину элемента, а margin как обычно.

Далее дадим задний фон для body

|  |
| --- |
| **body** {   background: #9adad8; } |

Чем отличается background от background-color?) В свойстве background мы можем указывать не только цвет!

Далее поработаем с контейнером, зададим width и height, также margin: auto, чтоб выровнять по центру ( на этом этапе можно сделать как удобно, не обязательно делать верстку абсолютно гибкой).

|  |
| --- |
| .game {  width: 50%;  height: 600px;  margin: auto;  position: relative;  display: flex;  flex-wrap: wrap;   } |

Но важно, дисплей флекс будет важен на этапе сборки. Нам нужен одномерный массив.

Сделаем расстояние между картами по 5px с каждой стороны. Чтоб можно было указать высоту и ширину карт в процентах нужно использовать функцию calc

|  |
| --- |
| .card {  width: calc(25% - 10px);  height: calc(33.3% - 10px);  margin: 5px;  } |

Добавим также position: relative, чтоб мы могли позиционировать элементы относительно родителя, cursor: pointer просто потому что это в тему.

|  |
| --- |
| .card {  width: calc(25% - 10px);  height: calc(33.3% - 10px);  margin: 5px;  cursor: pointer;  position: relative; } |

Делаем стили для самих картинок внутри дивов.

Далее все дело вашего вкуса, однако position: absolute должно быть обязательно, это свойство установит картинки одну поверх другой.

|  |
| --- |
| .front-face, .back-face {  width: 100%;  height: 100%;  padding: 20px;  position: absolute;  border-radius: 5px;  background: #876098 } |

Стили практически готовы, у нас уже есть тот рисунок, который присутствует на макете. Остальное мы просто не видим, угадай что это? Анимация

## Анимация

Давай сделаем эффект клика, чтоб наша картинка уменьшалась 3 секунды, когда мы на нее кликнем, в этом нам поможет псевдокласс :active.

За любые трансформации отвечает свойство transform, а за размер - значение scale, где 1 - это 100% размера, давай поставим 97%. Также есть прекрасное свойство transition, оно позволяет продлевать переход от одного состояния до другого.

|  |
| --- |
| .card:active {  transform: scale(0.97);  transition: transform .2s; } |

Смотрим какая получилась красота и продолжаем.

## Переворачиваем карты

Мы сделаем класс flip и будем присваивать его элементу, на который нажмут. Мы возьмем все элементы card с помощью querySelectorAll, повесим на них слушателя событий и будем тогглить класс flip по нажатию на карту.

|  |
| --- |
| **const** cards = document.querySelectorAll('.card');  **function** **flipCard**() {  event.target.parentElement.classList.toggle('flip'); }; cards.forEach(card => card.addEventListener('click', flipCard)); |

Теперь у нас добавляется и убирается класс по нажатию. Добавим стили этому классу

Познакомимся с новым свойством rotate

|  |
| --- |
| .card.flip {  transform: rotateY(180deg); } |

Теперь она можем при нажатии отображаться зеркально, как продлить переход от обычного положения до зеркального?

|  |
| --- |
| .card {  width: calc(25% - 10px);  height: calc(33.3% - 10px);  margin: 5px;  cursor: pointer;  position: relative;  transition: transform .5s; } |

Получилось очень круто, но мы видим только рубашку карты. Это происходит потому, что у нас две картинки наложены друг на друга, они абсолютно позиционированы. Исправляем. На самом деле карта и правду перевернулась, мы видим заднюю часть картинки, но ее можно спрятать с помощью свойства backface-visibility, по умолчанию оно visible, а нам надо hidden

|  |
| --- |
| .front-face, .back-face {  width: 100%;  height: 100%;  padding: 20px;  position: absolute;  border-radius: 5px;  background: #876098;  backface-visibility: hidden; } |

Но все еще ничего не происходит, потому что мы забыли перейти от 2d в трехмерную плоскость

|  |
| --- |
| .card {  width: calc(25% - 10px);  height: calc(33.3% - 10px);  margin: 5px;  cursor: pointer;  position: relative;  transition: transform .5s;  transform-style: preserve-3d; } |

Также есть очень интересная штука - perspective. Она свойство устанавливает, насколько далеко находится объект от пользователя. Чем ниже значение, тем больше эффект перспективы. Для тонкого эффекта давайте применим 1000px (можно и чуть меньше)

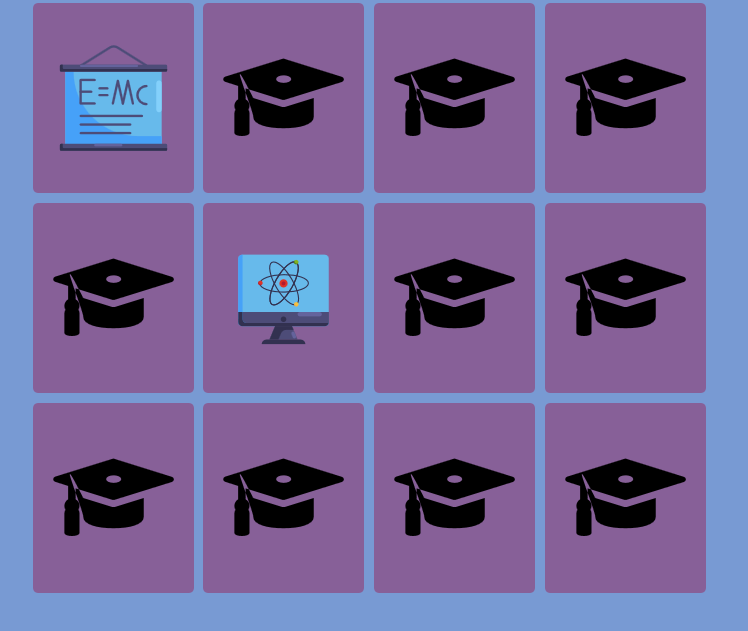
|  |
| --- |
| .game {  width: 50%;  height: 600px;  margin: auto;  display: flex;  flex-wrap: wrap;  perspective: 900px; } |

Карта прячется, кажется мы на верном пути. Но почему у нас оба изображения спрятались? Как это исправить? Поступим хитро, зададим катринке front-face изначальное положение развернутое

|  |
| --- |
| .front-face {  transform: rotateY(180deg); } |

## 

Желаемый эффект получен



Отлично, графическая составляющая готова, теперь самое “легкое” - логическая часть.

## Сравнение карт

Нам нужно перевернув одну карту ждать, пока мы перевернем вторую и если они не одинаковые - перевернуть обе обратно, а если они совпали - оставить и их нажать мы больше не сможем

Сделаем переменные для первой, второй карты и переменную, которая следит за тем, перевернуты ли они

|  |
| --- |
| **let** isFlippedCard = false;  **let** firstCard, secondCard; |

Совершаем проверку. Убираем тоггл, теперь он больше не нужен. Будем просто добавлять класс.

|  |
| --- |
| **function** **flipCard**() {  **let** item = event.target.parentElement   item.classList.add('flip');   **if** (!isFlippedCard) {  isFlippedCard = true;  firstCard = event.target.parentElement;  **return**;  } |

Теперь нам надо сравнивать каким то образом, что карты равны, есть много способов это сделать, мы применим один из них.

Всякий раз, когда мы хотим добавить дополнительную информацию к элементам HTML, мы можем использовать data-атрибуты. Используя следующий синтаксис: data-\* где, \* может быть любым словом, этот атрибут будет вставлен в свойство набора данных элемента. Итак, давайте добавим data-education к каждой карточке:

|  |
| --- |
| <div class="card" data-education="1">  <img class="front-face" src="img/1.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" data-education="1">  <img class="front-face" src="img/1.svg" />  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" data-education="2">  <img class="front-face" src="img/2.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" data-education="2">  <img class="front-face" src="img/2.svg"/>  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" data-education="3">  <img class="front-face" src="img/3.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" data-education="3">  <img class="front-face" src="img/3.svg" />  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" data-education="4">  <img class="front-face" src="img/4.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" data-education="4">  <img class="front-face" src="img/4.svg" />  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" data-education="5">  <img class="front-face" src="img/5.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" data-education="5">  <img class="front-face" src="img/5.svg" />  <img class="back-face" src="img/back.svg" />  </div>   <div class="card" data-education="6">  <img class="front-face" src="img/6.svg" />  <img class="back-face" src="img/back.svg" />  </div>  <div class="card" data-education="6">  <img class="front-face" src="img/6.svg" />  <img class="back-face" src="img/back.svg" />  </div> |

Уже ясно как она будет работать? Если даты равно то…

Если они будут равно, то нужно отключить их, а если нет - перевернуть обратно.

|  |
| --- |
| **function** **flipCard**() {  **let** item = event.target.parentElement  **if** (lockBoard) **return** lockBoard;  **if** (event.target.parentElement === firstCard) **return** firstCard;   item.classList.add('flip');   **if** (!isFlippedCard) {  isFlippedCard = true;  firstCard = event.target.parentElement;  **return**;  }  secondCard = event.target.parentElement;  firstCard.dataset.education === secondCard.dataset.education ? disableCards() : unflipCards(); } |

Давай напишем обе эти функции.

Чтоб отключить карты нам нужно сделать две вещи: убрать слушателей событий и обнулить переменные в которые все записывалось. Но так как сбрасывать переменную нам придется даже если карты совпали, нужно это вынести в отдельную функцию. Как “обнулить функцию”? Что выдаст переменная если мы ее объявим, а значения не укажем? Undefined. Это значит что в ней ничего нет, пустота. А как задать эту пустоту рукотворно? Для этого есть значение null, это тоже самое что undefined, но его никак нельзя получить не присвоив самостоятельно. Мы можем сделать isFlippedCard = undefined, но тогда это может привести к тому, что будет сложно выловить ошибку. Ведь undefined мы часто получаем при ошибках, когда код ничего не возвращает.

|  |
| --- |
| **function** **resetBoard**() {  isFlippedCard = false;  [firstCard, secondCard] = [null, null]; } |

И теперь выключаем слушателей

|  |
| --- |
| **function** **disableCards**() {  firstCard.removeEventListener('click', flipCard);  secondCard.removeEventListener('click', flipCard);  resetBoard(); } |

## Блокировка карт

Теперь сделаем функцию, которая будет переворачивать карты обратно. Также предлагаю сделать так, чтоб пока они переворачиваются мы не могли ни на что нажать. Введем переменную lockBoard. А функция будет ждать одну секунду, удалять класс flip и сбрасывать переменные карт.

А как код можно заставить запустить через секунду? Есть такая специальная функция, уже вшитая в язык программирования, называется setTimeout, в нее мы передаем два значения, первое - функция коллбэк (спросите у ребенка что такое коллбэк, если не знает - объясните), а второе через сколько милисекунд.

Соответственно нам нужно 1000.

|  |
| --- |
| **function** **flipCard**() {  **let** item = event.target.parentElement  **if (lockBoard) return lockBoard;**  **if** (event.target.parentElement === firstCard) **return** firstCard;   item.classList.add('flip');   **if** (!isFlippedCard) {  isFlippedCard = true;  firstCard = event.target.parentElement;  **return**;  }  secondCard = event.target.parentElement;  firstCard.dataset.education === secondCard.dataset.education ? disableCards() : unflipCards(); } |

|  |
| --- |
| **function** **unflipCards**() {  lockBoard = true;   setTimeout(() => {  firstCard.classList.remove('flip');  secondCard.classList.remove('flip');  resetBoard();  }, 1000); } |

## Великий рандом

Игра почти готова, осталось сделать рандом всех картинок. Мы из дивов сделаем массив, Есть замечательное свойство флекс элементов order, оно отвечает за порядковый номер элемента в флекс контейнере. Уже догадались?)

|  |
| --- |
| [...cards].forEach(card => {  **let** randomPos = Math.floor(Math.random() \* 12);  card.style.order = randomPos; }); |

## 

Все, игра готова, сможешь за 10 секунд открыть все карты?

## 

## Домашнее задание

1.Читать, смотреть youtube про setTimeout, отличие null от undefined, data-атрибуты, transform в css

2. Добавить попытки, будем тренировать память. Для более продвинутых сделать таймер.

|  |
| --- |
|  |