МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ

ГОУ ВО ЛНР "ЛУГАНСКИЙ ГОСУДАРСТВЕННЫЙ

УНИВЕРСИТЕТ ИМЕНИ ВЛАДИМИРА ДАЛЯ"

Кафедра "Автоматизация и компьютерно-интегрированные технологии"

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

по дисциплине "WEB технологии в системах автоматизации"

Тема: "Разработка игры Тетрис"

Студент Чередник. И.А.

(фамилия, инициалы) (подпись)

Группа ИТ-311

Защищено с оценкой

Руководитель работы асс. Сухов Д. А.

(фамилия, инициалы) (подпись)

# Луганск, 2022

Оглавление

[Введение. 3](#_Toc122282619)

[Проектирование HTML. 3](#_Toc122282620)

[CSS-стили. 4](#_Toc122282621)

[JS-скрипты. Работа тетриса. 6](#_Toc122282622)

[Заключение. 7](#_Toc122282623)

[Приложение. 8](#_Toc122282624)

[HTML: 8](#_Toc122282625)

[CSS: 9](#_Toc122282626)

[JS: 13](#_Toc122282627)

[Используемые источники. 19](#_Toc122282628)

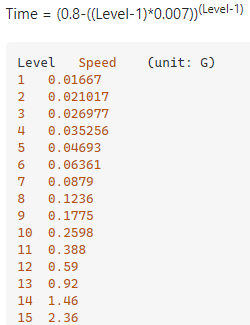
# Введение.

В данном проекте клиентская часть писалась в редакторе Visual Studio Code. Visual Studio Code — редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений.

За основу, в данном проекте взята популярная игра Tetris. Tetris — компьютерная игра, первоначально изобретённая и разработанная советским программистом Алексеем Пажитновым. Игра была выпущена 6 июня 1984 года — в это время Пажитнов работал в Вычислительном центре Академии наук СССР.

«Тетрис» представляет собой головоломку, построенную на использовании геометрических фигур «тетрамино» — разновидности полимино, состоящих из четырёх квадратов. Полимино в том или ином виде использовались в настольных играх и головоломках задолго до создания «Тетриса». Идею «Тетриса» Пажитнову подсказала игра в пентамино. Первоначальная версия игры была написана Пажитновым на языке программирования Паскаль для компьютера «Электроника-60». Коммерческая версия игры — первая из многих последующих — была выпущена американской компанией Spectrum HoloByte в 1987 году. В последующие годы «Тетрис» во множестве различных версий был портирован на великое множество устройств, включая всевозможные компьютеры и игровые консоли, а также такие устройства, как графические калькуляторы, мобильные телефоны, медиаплееры, карманные персональные компьютеры и — в качестве «пасхального яйца» — устройства, вовсе не предназначенные для воспроизведения медиаконтента, такие, как осциллограф и паяльник.

В разработанном аналоге тетриса, очки подсчитываются по схожей формуле и таблице:



# 

# Проектирование HTML.

Клиент представляет собой html-документ, который будет транслироваться каким-либо браузером.

Для того чтобы создать html-документ, достаточно создать текстовый файл, поменять его расширение с “.txt” на “.html” и в открытом файле с помощью любого текстового редактора вписать главный тег -“<html>тело</html>”. Это минимальные требования для создания html-документа.

Web-страницы могут существовать в любом формате, но в качестве стандарта принят Hyper Text Markup Language - язык разметки гипертекстов, предназначенный для создания форматированного текста, насыщенного изображениями, звуком, анимацией, видеоклипами и гипертекстовыми ссылками на другие документы, разбросанные как по всему Web-пространству, так и находящиеся на этом же сервере или являющиеся составной частью этого же проекта.

Работа с HTML – это способ усвоить особенности создания документов в стандартизированном языке, используя расширения, только если это необходимо.

Основным преимуществом HTML является то, что документ может быть просмотрен на WEB-браузерах различных типов и на различных платформах.

Документы могут быть созданы при помощи любого текстового редактора или специализированных HTML-редакторов и конвертеров. Выбор редактора, который будет использоваться для создания HTML-документов, зависит исключительно от понятия удобства и личных пристрастий каждого автора.

При написании клиентской части на языке HTML использовалась методология БЭМ (блок, элемент, модификатор). БЭМ предлагает общую семантическую модель для всех технологий, использующихся во frontend разработке (HTML, CSS, JavaScript, шаблоны и др.).

Используя понятия «блок», «элемент» и «модификатор» можно описать древовидную структуру документа. Такое описание называется BEM деревом и является семантическим представлением интерфейса, абстракцией над DOM деревом. Подобие такого дерева описано в данном проекте: смотреть приложение.

В данном проекте к html-документу подключаются отдельные файлы css-стилей и js-скриптов. Файлы стилей обычно подключают внутри служебного тега “<head>”, самый нижний подключенный файл в приоритете, то есть он перезаписывает те стили, которые были описаны в файлах выше. Js-файлы принято подключать перед закрывающим тегом “</body>”, это делается для того, чтобы перед выполнением скрипта успел под грузиться html-документ со стилями, а уже после началось выполнение кода скрипта. Файлы скрипта, с верху вниз наследуют описания переменных, функции, классы и т.д., это позволяет не переписывать один и тот же код несколько раз.

# CSS-стили.

Под термином CSS понимаем фразу “Каскадные Таблицы Стилей”. Они внедряются в HTML-код и не требуют никаких специальных редакторов и компиляторов, ведь интерпретируются оные обычным браузером.

Если формат HTML предоставляет информацию о составе документа, то таблицы стилей сообщают как он должен выглядеть. Таким образом каскадные таблицы стилей дают возможность хранить содержимое отдельно от его представления.

Термин «каскадный» означает, что в одной странице HTML могут использоваться разные стили. Браузер, поддерживающий таблицы стилей, будет следовать их порядку (как по каскаду), интерпретируя информацию стилей. Это означает, что вы можете использовать все три типа стилей, и броузер будет интерпретировать сначала связанные, затем внедренные и, наконец, встроенные стили. Даже если ко всему узлу будут применены образцы стилей, можно будет управлять отдельными аспектами страниц с помощью внедренных стилей, а отдельными областями внутри этих страниц - с помощью встроенных стилей.

Синтаксис таблицы стилей.

Таблицы стилей строятся в соответствии с определенным порядком (синтаксисом), в противном случае они не могут нормально работать.

Синтаксис всех методов, используемых для применения стилей к документу HTML, практически одинаков. Таблицы стилей составляются из определенных частей. Эти части включают следующие элементы:

• Указатель (Selector). Указатель является элементом, к которому будут применяться назначаемые вами атрибуты. Это может быть просто тег типа заголовка (H1) или абзаца (Р). Таблицы стилей позволяют использовать различные объекты, включая классы, которые будут кратко обсуждаться далее.

• Свойство (Property). Свойство определяет указатель. Например, если в качестве указателя выбран абзац, вы можете включить свойства, определяющие этот указатель. В свойства входят такие элементы, как поля, шрифты и фоновые изображения. В таблицах стилей существует много свойств, которые можно использовать для того, чтобы определить указатель.

• Значение (Value). Значения определяют свойства. Предположим, что у вас есть заголовок первого уровня H1 (указатель) и вы включаете свойство type family (семейство шрифта). Шрифт, который на самом деле будет применен к указанному фрагменту, задается значением этого свойства.

Существует три метода для применения таблицы стилей к документу HTML:

• Встроенный (Inline). Этот метод позволяет взять любой тег HTML и добавить к нему стиль. Использование встроенного метода предоставляет максимальный контроль над всеми свойствами Web-страницы. Предположим, вы хотите задать внешний вид отдельного абзаца. Вы можете просто добавить атрибут style к тегу абзаца, и броузер отобразит этот абзац с помощью параметров стиля, добавленного в код.

• Внедренный (Embedded). Внедрение позволяет контролировать всю страницу HTML. При использовании тега <style>, помещенного внутри раздела <head> страницы HTML, в код вставляются детализированные атрибуты стиля, которые будут применяться ко всей странице.

• Связанный (Linked или External). Связанная таблица стилей - мощный инструмент, который позволяет создавать образцы стилей (master styles), которые можно затем применять ко всему узлу. Основной документ таблицы стилей (расширение.css) создается Web-дизайнером. Этот документ содержит стили, которые будут едиными для всего Web-узла (неважно, содержит одну страницу или тысячи страниц). Любая страница, связанная с этим документом, будет использовать указанные стили.

В приведенном проекте html-документ использует “Связанный” стиль, то есть стили написаны в отдельных файлах, и подключены к html-документу. Все модификаторы описаны в файле “style.css”, второй файл – “reset.css”, необходим для обнуления стилей страницы, значения которых уже стоят по умолчанию.

Как видно, в главном HTML-файле блоки и элементы являются только описанными и не имеют явного содержимого, основные свойства для удобства будут описаны в отдельных файлах.

Для позиционирования большинства элементов используется технология flexbox. Flexbox разработан для оптимизации раскладки элементов. Он облегчает выравнивание элементов по горизонтали и вертикали, смену направления и порядок отображения элементов, позволяет растягивать блоки на всю высоту родителя или опускать их к нижнему краю. В остальных случаях применяется абсолютное позиционирование.

Адаптивная верстка отсутствует за ненадобностью.

В стили так же импортированы особые шрифты с использованием @font-face. Важно понимать: @font-face это не подключение файла шрифта буквально, а описание конкретного набора всех вышеупомянутых свойств для загруженного файла, до которого можно достучаться через обращение к ним же в правилах CSS нужных элементов.

В проекте использовались два вида единиц измерения, такие как “px” и “%”, а также в некоторых случаях за ненадобностью приведенных выше единиц измерения было использовано такое ключевое слово, как auto. Используемые единицы измерения в проекте. Auto – это сокращение от слова automatic (с англ. языка переводится как автоматический). Само определение говорит само за себя. Если задавать его свойствам CSS, то браузер будет автоматически подбирать нужное значение для свойства. Как правило, если свойству не задано какое-то конкретное значение, то оно автоматически будет находится в значении auto.

Дизайн, описываемый в стилях достаточно незамысловатый, цветовая палитра черно-белая, шрифты в ретро-пиксельном стиле, фигуры в одном цвете. Такие нехитрые стили описаны для того, чтобы уменьшить нагрузку на сервер. Стили смотреть в приложении.

# JS-скрипты. Работа тетриса.

JavaScript – это язык управления сценариями просмотра гипертекстовых страниц Web на стороне клиента. Если быть более точным, то JavaScript – это не только язык программирования на стороне клиента. Liveware, прародитель JavaScript, является средством подстановок на стороне сервера Netscape. Однако наибольшую популярность JavaScript обеспечило программирование на стороне клиента.

Основная идея JavaScript состоит в возможности изменения значений атрибутов HTML-контейнеров и свойств среды отображения в процессе просмотра HTML-страницы пользователем. При этом перезагрузки страницы не происходит.

JavaScript стандартизован ECMA (European Computer Manufacturers Association - Ассоциация европейских производителей компьютеров). Соответствующие стандарты носят названия ECMA-262 и ISO-16262. Этими стандартами определяется язык ECMAScript, который примерно эквивалентен JavaScript 1.1.

Часть проекта, написанная на JavaScript, использует ECMAScript 6.

• В скрипте описаны основные действия игрока, такие как автоматическое и управляемое клиентом движение вниз, по сторонам и вращение фигуры, а также функции подсчета очков, конца игры и т.д.

Все действия основаны на изменении модификаторов элементов в определенном порядке.

В начале игровое поле заполняется пустыми ячейками, генерируется случайная фигура. Фигуры представляют собой массив из четырех чисел, являющиеся индексами ячеек на игровом поле, которые будут вместе образовывать фигуру. Далее запускается интервал, который опускает фигуру вниз через определенный промежуток времени, этот промежуток постепенно уменьшается с увеличением количества убранных игроком линий.

Пока идет игра, игрок может двигать фигуру, само движение представляет изменение модификаторов ячеек. Движение осуществляются при помощи DOM-событий. Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM). В проекте использовался только один обработчик событий, keydown, повешенный на документ, он обрабатывает любые нажатия клавиш с клавиатуры, которые в будущем будут проверяться через метод which (Примечание: данный метод является устаревшим, рекомендуется использовать метод code).

Поворот фигуры так же обрабатывается обработчиком событий. Описанный в скрипте поворот осуществляется при помощи поворота матрицы, из преобразованных индексов ячеек фигуры, на девяносто градусов. После поворота идет сортировка массива фигуры по определенному порядку. Сам отсортированный массив представляет собой массив, десятые и сотые части которого отсортированы по убыванию, а единичные части, по возрастанию.

# Заключение.

Язык Web-программирования JavaScript приобрел в последнее время большую популярность. Он может быть использован не только для управления веб-страницами, а также в качестве средства решения ряда задач. Исходя из проведенных исследований, можно сделать вывод: JavaScript может многое – например, с его помощью можно без труда выполнять вычисления различных функций, проводить вычисления математических выражений и т.д., не затрачивая на это большое количество времени.

# Приложение.

## HTML:

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

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

<title>Tetris</title>

<link rel="stylesheet" href="styles/reset.css">

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

</head>

<body>

<video autoplay muted loop id="myVideo">

<source src="gif/Thriller\_low\_quality.mp4" type="video/mp4">

</video>

<div class="wrapper">

<div class="player\_\_main">

<div class="player\_\_main-inf">

<div class="player\_\_main-score"></div>

<div class="player\_\_main-line">Line: <span class="player\_\_main-count\_\_line"></span></div>

<div class="player\_\_main-nextTetrominos"></div>

</div>

<div class="player\_\_main-playZone"></div>

<div class="player\_\_main-name">PLAYER 1</div>

</div>

<div class="player\_\_first">

<div class="player\_\_first-inf">

<div class="player\_\_first-score"></div>

<div class="player\_\_first-line">Line: <span class="player\_\_first-count\_\_line"></span></div>

<div class="player\_\_first-nextTetrominos"></div>

</div>

<div class="player\_\_first-playZone"></div>

<div class="player\_\_first-name">PLAYER 2</div>

</div>

</div>

<script src="scripts/main\_player.js"></script>

<script src="scripts/other\_player.js"></script>

<script src="scripts/connect.js"></script>

</body>

</html>

## CSS:

#myVideo {

position: fixed;

left: 50%;

top: 50%;

transform: translate(-50%, -50%);

min-width: 80%;

min-height: 80%;

z-index: 1;

}

body {

max-width: 100%;

max-height: 100%;

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

background-color: #000;

}

.wrapper {

display: flex;

flex-direction: row;

justify-content: space-evenly;

align-items: center;

width: 100%;

max-height: 100%;

z-index: 2;

}

.player\_\_main, .player\_\_first {

display: flex;

justify-content: space-evenly;

align-items: center;

flex-direction: column;

border: 1px solid #c8c8c8;

background-color: #000;

width: 300px;

height: auto;

position: relative;

}

.player\_\_main-inf, .player\_\_first-inf {

display: flex;

flex-direction: row;

flex-wrap: wrap;

justify-content: space-evenly;

align-items: center;

border-bottom: 1px solid #c8c8c8;

background-color: #000;

position: absolute;

top: 0px;

width: 100%;

height: auto;

padding: 5px;

}

.player\_\_main-score, .player\_\_first-score {

display: flex;

flex-direction: row;

justify-content: center;

align-items: center;

flex-wrap: wrap;

width: 300px;

height: 40px;

font-family: "Pixel";

font-size: 20px;

color: #c8c8c8;

padding: 5px;

}

.player\_\_main-line, .player\_\_first-line {

display: flex;

flex-direction: row;

flex-wrap: wrap;

justify-content: center;

align-items: center;

width: 120px;

height: 35px;

font-family: "Pixel";

color: #c8c8c8;

padding: 5px;

}

.player\_\_main-nextTetrominos, .player\_\_first-nextTetrominos {

display: flex;

flex-direction: row;

flex-wrap: wrap;

width: 60px;

height: 60px;

padding: 5px;

}

.player\_\_main-nextTetrominos-cell, .player\_\_first-nextTetrominos-cell {

display: flex;

flex-wrap: wrap;

flex: 1 0 25%;

height: 25%;

border-radius: 15%;

}

.player\_\_main-playZone, .player\_\_first-playZone {

display: flex;

flex-wrap: wrap;

width: 300px;

height: 680px;

margin-bottom: 25px;

margin-top: 50px;

padding: 0px 2px;

}

.player\_\_main-name, .player\_\_first-name {

display: flex;

justify-content: center;

align-items: center;

position: absolute;

bottom: 0;

width: 100%;

height: 6.1%;

font-family: "Pixel";

font-size: 20px;

color: #c8c8c8;

background-color: #000;

border-top: 1px solid #c8c8c8;

}

.other\_\_players {

display: flex;

width: 300px;

height: 600px;

border: 1px solid #c8c8c8;

}

.tetrominos\_\_empty-cell {

display: flex;

flex-wrap: wrap;

flex: 1 0 10%;

height: 30px;

background-color: #000;

}

.tetrominos\_\_move-cell, .tetrominos\_\_fill-cell{

display: flex;

flex-wrap: wrap;

flex: 1 0 10%;

height: 30px;

color: #000;

border-radius: 15%;

border: 8px solid #c8c8c8;

}

.gameOverPopup {

display: flex;

justify-content: center;

align-items: center;

z-index: 1;

width: 100%;

border: 1px solid #c8c8c8;

height: 10%;

font-size: 28px;

font-family: "Pixel";

color: #c8c8c8;

background-color: #000;

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

letter-spacing: 5px;

}

/\* FONTS \*/

@font-face {

font-family: "Pixel";

src: url("../fonts/ka1.ttf") format("truetype");

font-style: normal;

font-weight: normal;

}

## 

## JS:

let tetrominosSpeedDown = 600;

let gameStatus = true;

let countPlayerScore = 0;

let countPlayerLines = 0;

let nextTetrominos = null;

let currentTetrominos = null;

const playZone = document.querySelector(".player\_\_main-playZone");

const nextTetrominosZone = document.querySelector(".player\_\_main-nextTetrominos");

const playerScore = document.querySelector(".player\_\_main-score");

const playerLines = document.querySelector(".player\_\_main-count\_\_line");

const tetrominos = [

[3, 4, 5, 6],

[14, 15, 16, 4],

[14, 15, 16, 6],

[14, 15, 4, 5],

[14, 15, 5, 6],

[15, 16, 4, 5],

[14, 15, 16, 5],

];

for (let row = 0; row < 23; row++) {

for (let col = 0; col < 10; col++) {

let emptyCell = document.createElement("div");

emptyCell.classList.add("tetrominos\_\_empty-cell");

playZone.appendChild(emptyCell);

};

};

for (let row = 0; row < 4; row++) {

for (let col = 0; col < 4; col++) {

let emptyCell = document.createElement("div");

emptyCell.classList.add("player\_\_main-nextTetrominos-cell");

nextTetrominosZone.appendChild(emptyCell);

};

};

let playElems = document.querySelectorAll(".tetrominos\_\_empty-cell");

let nextElems = document.querySelectorAll(".player\_\_main-nextTetrominos-cell");

for (let i = playElems.length - 10; i < playElems.length; i++) {

playElems[i].className = "tetrominos\_\_fill-cell";

}

function getRandomNumber(min, max) { return Math.floor(Math.random() \* (max - min + 1)) + min;};

function CreateTetrominos(tetrominos) {

let num = getRandomNumber(0, 6);

let tetromino = [];

for(let i = 0; i < tetrominos[num].length; i++) {

playElems[tetrominos[num][i]].className = "tetrominos\_\_move-cell";

tetromino[i] = tetrominos[num][i];

}

return tetromino;

};

function CheckTetrominos(arr, interval) {

for (let i = 0; i < 4; i++) {

if (playElems[arr[i] + 10].className == "tetrominos\_\_fill-cell") {

clearInterval(interval);

setTimeout(() => {

for (let j = 0; j < 4; j++) playElems[arr[j]].className = "tetrominos\_\_fill-cell";

if (gameStatus) {

CheckDestroyLine();

currentTetrominos = nextTetrominos;

nextTetrominos = CreateTetrominos(tetrominos);

ShowInfo(nextTetrominos, countPlayerLines, countPlayerScore);

TetrominosAutoDown(currentTetrominos);

}

}, 10);

break;

}

};

}

function CheckGameStatus(interval) {

setTimeout(() => {

for (let i = 20; i <= 29; i++) {

if (playElems[i].className == "tetrominos\_\_fill-cell") {

clearInterval(interval);

gameStatus = false;

EndGame();

break;

}

}

});

}

function TetrominosAutoDown(arr) {

let intervalDownwardMovement = setInterval(() => {

CheckGameStatus(intervalDownwardMovement);

for(let i = 0; i < 4; i++) {

playElems[arr[i]].className = "tetrominos\_\_empty-cell";

arr[i] += 10;

playElems[arr[i]].className = "tetrominos\_\_move-cell";

}

CheckTetrominos(arr, intervalDownwardMovement);

}, tetrominosSpeedDown);

MoveTetrominos(arr);

};

function MoveTetrominos(arr) {

document.addEventListener("keydown", (event) => {

if (event.which === 37 && playElems[arr[2]].className == "tetrominos\_\_move-cell") {

mySort(arr);

let flag = true;

for (let i = 0; i < 4; i++) {

if ((arr[i]) % 10 == 0 || playElems[arr[i] - 1].className == "tetrominos\_\_fill-cell") flag = false;

if (arr[i] < 20 || playElems[arr[i] + 10 - 1].className == "tetrominos\_\_fill-cell") flag = false;

}

if (flag) {

for (let i = 4 - 1; i >= 0 ; i--) {

playElems[arr[i]].className = "tetrominos\_\_empty-cell";

arr[i] -= 1;

playElems[arr[i]].className = "tetrominos\_\_move-cell";

}

}

} else if (event.which === 39 && playElems[arr[2]].className == "tetrominos\_\_move-cell") {

mySort(arr);

let flag = true;

for (let i = 0; i < 4; i++) {

if ((arr[i]) % 10 == 9 || playElems[arr[i] + 1].className == "tetrominos\_\_fill-cell") flag = false;

if (arr[i] < 20 || playElems[arr[i] + 10 + 1].className == "tetrominos\_\_fill-cell") flag = false;

}

for (let i = 0; i < 4; i++) {

if (flag) {

playElems[arr[i]].className = "tetrominos\_\_empty-cell";

arr[i] += 1;

playElems[arr[i]].className = "tetrominos\_\_move-cell";

}

}

} else if (event.which === 40 && playElems[arr[2]].className == "tetrominos\_\_move-cell") {

mySort(arr);

let flag = true;

for (let i = 0; i < 4; i++)

if (arr[i] >= 210 || playElems[arr[i] + 20].className == "tetrominos\_\_fill-cell")

flag = false;

if (flag) {

for (let i = 0; i < 4 ; i++) {

playElems[arr[i]].className = "tetrominos\_\_empty-cell";

arr[i] += 10;

playElems[arr[i]].className = "tetrominos\_\_move-cell";

}

}

} else if (event.which === 38 && playElems[arr[2]].className == "tetrominos\_\_move-cell") {

let matrix = [];

let oldMatrix = [];

let arrCount = 0;

let flag = true;

mySort(arr);

let start = arr[1];

for (let row = start - 22, i = 0; row < start + 22, i < 5; row += 10, i++) {

matrix[i] = [];

for (let col = row, j = 0; col < row + 5, j < 5; col++, j++) {

matrix[i][j] = col;

}

}

oldMatrix = matrix;

matrix = matrix.map((row, i) => row.map((val, j) => matrix[4 - j][i]));

for (let row = start - 22, i = 0; row < start + 22, i < 5; row += 10, i++) {

for (let col = row, j = 0; col < row + 5, j < 5; col++, j++) {

if (playElems[matrix[i][j]].className == "tetrominos\_\_move-cell") {

if (playElems[oldMatrix[i][j]].className == "tetrominos\_\_fill-cell") flag = false;

else if (matrix[i][j] % 10 > 0 && col % 10 == 0) flag = false;

else if (matrix[i][j] % 10 == 0 && col % 10 == 1) flag = false;

else if (col > 200) flag = false;

else if (col < 20) flag = false;

else if (playElems[col + 10].className == "tetrominos\_\_fill-cell") flag = false;

}

}

}

if (flag) {

for (let row = start - 22, i = 0; row < start + 22, i < 5; row += 10, i++) {

for (let col = row, j = 0; col < row + 5, j < 5; col++, j++) {

if (playElems[matrix[i][j]].className == "tetrominos\_\_move-cell") {

if (col != matrix[i][j]) {

playElems[matrix[i][j]].className = "tetrominos\_\_empty-cell";

arr[arrCount] = col;

arrCount++;

} else {

playElems[col].className = playElems[matrix[i][j]].className;

arr[arrCount] = col;

arrCount++;

}

}

}

}

for (let i = 0; i < 4; i++) {

playElems[arr[i]].className = "tetrominos\_\_move-cell";

}

mySort(arr);

} else {

return;

}

}

});

}

function mySort(arr) {

arr.sort((a, b) => {

return b - a;

});

let resultArr = [];

let count = 0;

let row = 0;

resultArr[row] = [];

for (let i = 0; i < 4; i++) {

if (Math.trunc(Number(String(arr[i] / 10))) > Math.trunc(Number(String(arr[i + 1] / 10)))) {

resultArr[row][count] = arr[i];

row++;

resultArr[row] = [];

count = 0;

} else {

resultArr[row][count] = arr[i];

count++;

if (count > 4) return;

}

}

arr = [];

count = 0;

for(let i = 0; i <= row; i++) {

resultArr[i] = resultArr[i].sort((a, b) => {return a - b;});

for (let j = 0; j < resultArr[i].length; j++) {

arr[count] = resultArr[i][j];

count++;

}

}

}

function ShowInfo(nextTetrominos, line, score) {

for(let i = 0; i < nextElems.length; i++) nextElems[i].style.border = "none";

for(let i = 0; i < nextTetrominos.length; i++) {

if (nextTetrominos[i] > 10) nextElems[nextTetrominos[i] - 5].style.border = "4px solid #c8c8c8";

else nextElems[nextTetrominos[i] + 1].style.border = "4px solid #c8c8c8";

}

playerLines.innerHTML = line;

playerScore.innerHTML = score;

}

function CheckDestroyLine() {

let countLines = 0;

for(let i = 0; i < (playElems.length - 10) / 10; i++) {

let isLine = true;

for (let j = i \* 10; j < (i \* 10) + 10; j++) {

if (playElems[j].className != "tetrominos\_\_fill-cell") {

isLine = false;

break;

}

}

if (isLine) {

for (let j = i \* 10; j < (i \* 10) + 10; j++) {

playElems[j].className = "tetrominos\_\_empty-cell";

}

for (let j = (i \* 10); j > 0; j--) {

if (playElems[j].className == "tetrominos\_\_fill-cell") {

playElems[j].className = "tetrominos\_\_empty-cell";

playElems[j + 10].className = "tetrominos\_\_fill-cell";

}

}

countLines++;

if (countPlayerLines < 151) {

tetrominosSpeedDown -= 3;

}

}

}

if (countLines) {

countPlayerLines += countLines;

countPlayerScore += ((countLines \* 2) + 1) \* 100;

}

}

function Start(gameStatus) {

currentTetrominos = CreateTetrominos(tetrominos);

nextTetrominos = CreateTetrominos(tetrominos);

ShowInfo(nextTetrominos, countPlayerLines, countPlayerScore);

TetrominosAutoDown(currentTetrominos);

}

function EndGame() {

let gameOverWindowPopup = document.createElement("div");

gameOverWindowPopup.className = "gameOverPopup";

gameOverWindowPopup.innerHTML = "GAME OVER";

document.querySelector(".player\_\_main").appendChild(gameOverWindowPopup);

}

Start(gameStatus);

# Используемые источники.

<https://learn.javascript.ru/>

<https://www.w3schools.com/css/>

<http://htmlbook.ru/>

<https://en.wikipedia.org/wiki/Tetris>