

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе № 1
по дисциплине «Web-технологии»
Тема: Тетрис на JavaScript.

Студент гр. 0382

Корсунов А.А.

Преподаватель

Беляев С.А.

Санкт-Петербург

2022

1. Цель работы.

Целью работы является изучение работы web-сервера nginx со статическими файлами и создание клиентских JavaScript web-приложений.

2. Основные теоретические сведения.

Асимметричные ключи используются в асимметричных алгоритмах шифрования и являются ключевой парой. Закрытый ключ известен только владельцу. Открытый ключ может быть опубликован и используется для проверки подлинности подписанного документа (сообщения). Открытый ключ вычисляется, как значение некоторой функции от закрытого ключа, но знание открытого ключа не дает возможности определить закрытый ключ.

3. Постановка задачи.

Задачи:

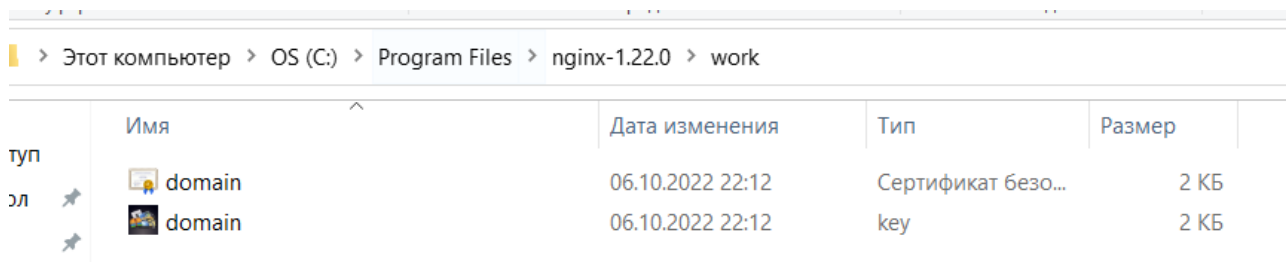
- 1) Генерация открытого и закрытого ключей для использования шифрования;
- 2) Настройка сервера nginx для работы по протоколу HTTPS;
- 3) Разработка интерфейса web-приложения;
- 4) Обеспечение ввода имени пользователя;
- 5) Обеспечение создания новой фигуры для тетриса по таймеру и её движение;
- 6) Обеспечение управления пользователем падающей фигурой;
- 7) Обеспечение исчезновения ряда, если он заполнен;
- 8) По окончании игры – отображение таблицы рекордов, которая хранится в браузере пользователя;

4. Ход работы.

1) Генерация открытого и закрытого ключей для использования шифрования.

Были созданы 2 ключа:

```
openssl req -newkey rsa:2048 -nodes -keyout domain.key  
-x509 -days 365 -out domain.crt
```





Этот компьютер > OS (C:) > Program Files > nginx-1.22.0 > work				
тип	Имя	Дата изменения	Тип	Размер
эл	 domain	06.10.2022 22:12	Сертификат безо...	2 КБ
	 domain	06.10.2022 22:12	key	2 КБ

Рисунок 1 - закрытый и открытый ключи

2) Настройка сервера nginx для работы по протоколу HTTPS..

Было установлено nginx/1.14.0 на систему «Windows 10» и настроен по протоколу HTTPS

```
server{  
    listen 443 ssl http2;  
    #listen 80;  
    server_name localhost;  
  
    ssl_certificate      "../work/domain.crt";  
    ssl_certificate_key  "../work/domain.key";  
  
    ssl_session_cache    shared:SSL:1m;  
    ssl_session_timeout  5m;  
  
    ssl_ciphers  HIGH:!aNULL:!MD5;  
    ssl_prefer_server_ciphers  on;  
  
    location / {  
        root "C:\WEBBBB\Work\work\data\www";  
    }  
}
```

Рисунок 2 — конфигурация nginx.conf

3) Разработка интерфейса web-приложения.

Интерфейс приложения состоит из страниц `index.html` и `main.html`. `Index.html` содержит ввод имени пользователя (рис. 3). `Main.html` содержит саму игру и требуемые по заданию данные (рис. 4);

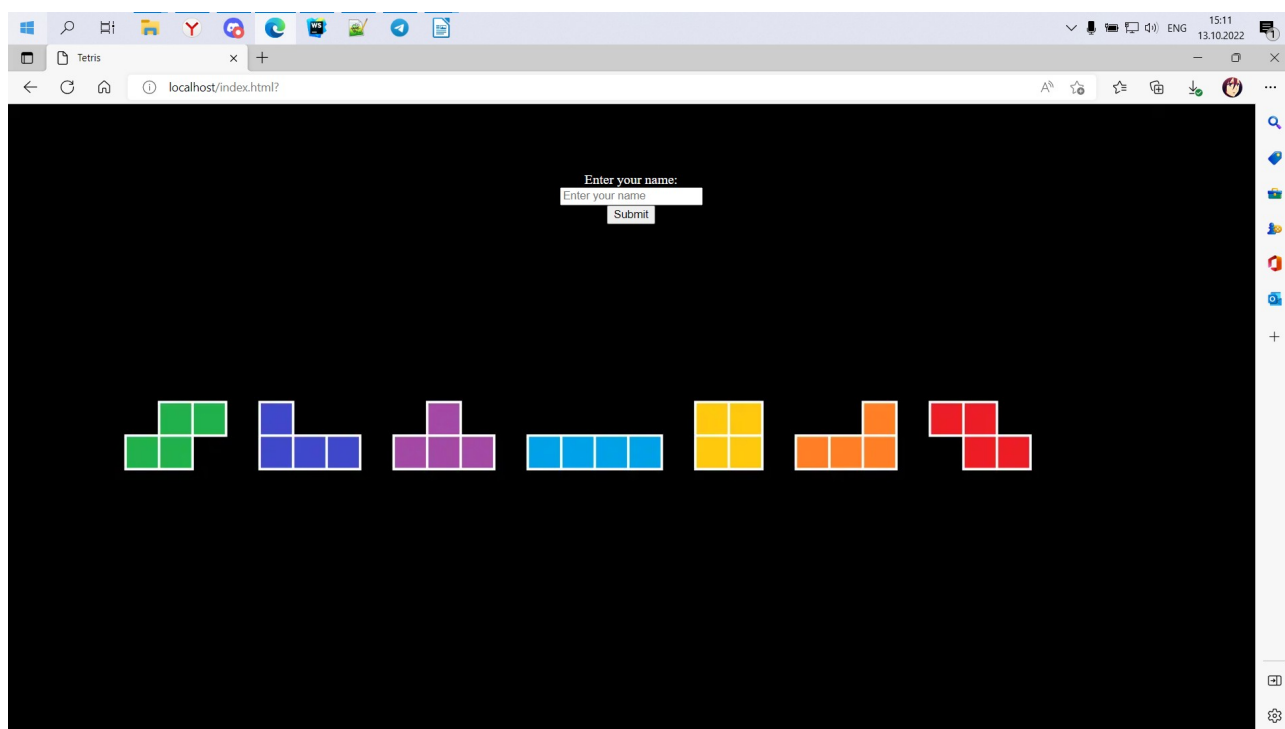


Рисунок 3 — `index.html`

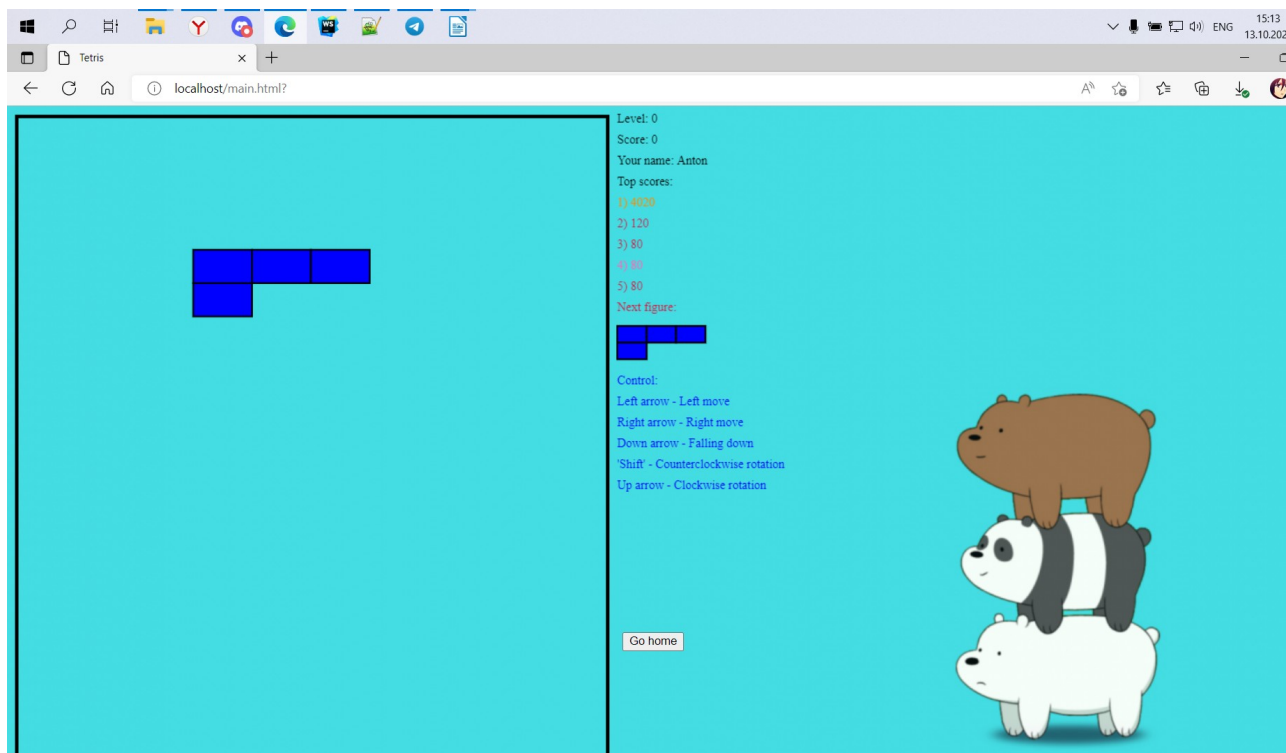


Рисунок 4 - main.html

4) Обеспечение ввода имени пользователя.

Пользователь вводит свое имя в специальную форму (рис.1), затем нажимает кнопку «Submit», после нажатия которой пользователь перенаправляется на страницу с игрой. Имя пользователя сохраняется в локальное хранилище данных, для того чтобы вывести его на странице с игрой. Последнее введенное имя пользователя отображается на страницу входа в специальной форме.

Работа с локальным хранилищем происходит в index.html благодаря двум функциям: store() и read(), первая — помещает в хранилище имя пользователя, а вторая извлекает имя из хранилища;

```

<script>  --
function store() {
    if (document.getElementById('name_input').value === null) {
        alert("Enter the name!");
    } else
        localStorage.setItem('#tetris.name_input', document.getElementById('name_input').value);
}

function read() {
    if (localStorage.hasOwnProperty("#tetris.name_input"))
    {
        let name = localStorage.getItem("#tetris.name_input");
        let input = document.getElementById("name_input");
        input.value = name;

        return name
    }
}
</script>

```

Рисунок 5 — функции сохранения и считывания имени пользователя

5) Обеспечение создания новой фигуры для тетриса по таймеру и её движение.

Вся логика игры происходит в `game.js`: каждая фигура есть объект, у которого имеется двумерный массив, в котором значение «0» означает свободное пространство, а цифры от «1» до «7» принадлежность к той или иной фигуре, а также одну из клеток самой фигуры (рис. 6). Новая фигура создается (метод `create_next_figure_move`) при старте и рестарте игры, а также когда «активная» фигура стала частью поля. Активная фигура — текущая падающая в «стакан» фигура. Активная фигура не является частью поля (стакана) до того момента, пока не упадет на дно, либо же на другую фигуру (которая к тому моменту уже является частью поля). Важно отметить, что хоть фигура и не является частью поля в момент падения, она отображается на странице — это обуславливается тем, что у каждой фигуры есть координата левого верхнего угла своего массива в массиве поля, а также методом `get_field()`, в котором создается текущая копия поля и в нее записывается падающая фигура;

Движение фигуры реализовано с помощью добавления обработчиков событий (addEventListener) в классе controller. Класс controller создан для управления логикой и графикой в совокупности. В конструкторе этого класса происходит добавление обработчиков событий, для которых были написаны два метода key_up_uses, которое содержит единственный случай — когда отпустить клавишу «стрелка вниз» происходит остановка таймера (для того, чтобы фигура не могла двигаться сама по себе, когда пользователь хочет, чтобы она упала вниз), и key_down_uses – перемещение фигуры на три допустимых стороны (влево, вправо, вниз), повороты на 90 градусов влево и вправо, а также случай, когда игра завершилась — пользователю предлагается начать ее заново. Таймер реализован в методах start_timer (рис. 7) – вычисляет интервалы падения фигуры (изначально — 1000 мл, минимально — 100 мл), задается интервал с помощью метода setInterval, stop_interval – останавливает интервал и сбрасывает текущее время интервалов.

```
figure_next.blocks =  
[  
    [0,0,0,0],  
    [1,1,1,1],  
    [0,0,0,0],  
    [0,0,0,0],  
];  
break;  
case 'J':  
    figure_next.blocks =  
    [  
        [0,0,0],  
        [2,2,2],  
        [0,0,2]  
    ];
```

Рисунок 6 — представление фигуры в логике

```

stop_timer()
{
    if (this.interval_id)
    {
        clearInterval(this.interval_id);
        this.interval_id = null;
    }
}

start_timer()
{
    const speed = 1000 - this.game.get_field().level * 100; //скорость падения

    if (!this.interval_id)
    {
        this.interval_id = setInterval( handler: () => {
            this.update_timer();
        }, timeout: speed > 0? speed: 100);
    }
}

```

Рисунок 7 — работа с таймером

6) Обеспечение управления пользователем падающей фигурой.

Управление падающей фигуры частично описано в пункте 5. Дополнительно можно отметить, что на уровне логики (в классе game) присутствует метод `is_figure_out_of_bounds`, который возвращает `true`, если фигура вышла за границы поля (по координатам падающей фигуры), или столкнулась с другой фигурой (как уже было описано выше — с частью поля). Данный метод используется в методах перемещения и поворота фигуры;


```

is_figure_out_of_bounds()
{
    for (let y = 0; y < this.figure_move.blocks.length; y++)
    {
        for (let x = 0; x < this.figure_move.blocks[y].length; x++)
        {
            if (this.figure_move.blocks[y][x] !== 0 && //если в поле фигуры 0, то все равно на пределы поля игры и другие фигуры
                (this.field[this.figure_move.y + y] === undefined ||
                 this.field[this.figure_move.y + y][this.figure_move.x + x] === undefined ||
                 this.field[this.figure_move.y + y][this.figure_move.x + x] !== 0))
            {
                return true;
            }
        }
    }
    return false;
}

```

Рисунок 8 — проверка на выход за поле и столкновение с другими фигурами

```

switch (event.keyCode) //keyCode возвращает числовой код клавиши
{
    case 37: //НАЛЕВО(стрелка)
        this.game.move_figure_left();
        this.update_view();
        break;
    case 38: //ВВЕРХ(стрелка)
        this.game.rotate_figure();
        this.update_view();
        break;
    case 39: //ВПРАВО(стрелка)
        this.game.move_figure_right();
        this.update_view();
        break;
    case 40: //ВНИЗ(стрелка)
        this.stop_timer();
        this.game.move_figure_down();
        this.update_view();
        break;
    case 16: //SHIFT
        this.game.rotate_figure_left();
        this.update_view();
        break;
    case 13: //ENTER
        if (state.is_game_over)
        {
            this.restart();
        }
}

```

Рисунок 9 — пользовательское управление фигурой

7) Обеспечение исчезновения ряда, если он заполнен.

Удаление ряда реализовано в классе `game` в методе `delete_lines()`. Создается массив (изначально пустой), в который записываются индексы полностью заполненных строк. После обхода поля из поля удаляются все строки по индексам из этого массива, а в начало массива поля (т. е. в начало поля) добавляются пустые строки. Важно отметить, что перебор массива происходит в обратном порядке и индексы в массива записываются в начало, а

не в конец — это сделано для того, чтобы при удалении из массива заполненных строк не были потеряны некоторые индексы строк, которые могут быть нужны в ходе перебора (т. е. чтобы не сбились индексы в самом массиве).

```
delete_lines()
{
    const rows = 20;
    const columns = 10;
    let lines = []; //массив строк, которые нужно удалить

    for (let y = rows - 1; y >= 0; y--)
    {
        let blocks_count = 0; //количество непустых блоков на линии (строке)
        for (let x = 0; x < columns; x++)
        {
            if (this.field[y][x] !== 0)
            {
                blocks_count++;
                //console.log(blocks_count);
            }
        }
        if (blocks_count === 0) //если линия пуста, то блоков сверху нет
        {
            break;
        } else if (blocks_count < columns) //если линия не пуста, но полностью не заполнена
        {
            continue;
        } else //оставшийся вариант - линия заполнена
        {
            lines.unshift(y); //добавим индекс строки в начало
        }
    }
}
```

Рисунок 10 — удаление строк

```
for (let i of lines)
{
    this.field.splice(i, deleteCount: 1); // удаляем заполненную линию
    this.field.unshift(new Array(columns).fill( value: 0)); //добавляем пустую линию сверху поля
}
//console.log((lines.length));
return lines.length;
```

Рисунок 11 — продолжение рисунка 10

8) По окончании игры – отображение таблицы рекордов, которая хранится в браузере пользователя;

Таблица рекордов — массив из пяти элементов, в котором записаны максимальные очки пользователей. Изначально таблица пуста (заполнена нулями), при начале игры из локального хранилища достаются данные таблицы (если они есть), в конце игры последний элемент таблицы (т. к. таблицы отсортирована от большего к меньшему) сравнивается с набранными очками и добавляется в таблицу (если они больше), после чего таблицы опять сортируется. Таблицы выводятся как в течении игры, так и при экране конца игры.

```
make_best_scores()
{
    this.best_scores.pop();
    this.best_scores.push(this.score);
    this.best_scores.sort( compareFn: function (a : number , b : number ){return b - a});
    localStorage.setItem("best_scores", JSON.stringify(this.best_scores));
}

table_scores(best_scores)
{
    if (localStorage.getItem( key: "best_scores") === null || localStorage.getItem( key: "best_scores") === undefined)
    {
        localStorage.setItem("best_scores", JSON.stringify(best_scores));
    }
    return JSON.parse(localStorage.getItem( key: "best_scores"));
}
```

Рисунок 12 — реализация таблицы рекордов

5. Функции и структуры данных:

Класс game

- make_best_scores – добавляет в таблицу рекордов набранные очки;
- table_scores – возвращает из локального хранилища таблицу рекордов;

- `restart` – обнуляет все значения для рестарта игры;
- `update_score` – обновляет текущие очки в зависимости от собранных линий;
- `delete_lines` – удаляет заполненные линии;
- `create_next_figure_move` – случайным образом меняет текущую активную фигуру на одну из семи заранее созданных;
- `get_field` – возвращает данные копии поля, в которое вписана фигура;
- `create_field` – создает новое поле;
- `rotate_figure_left` – поворачивает фигуру против часовой стрелки на 90 градусов;
- `rotate_figure` – поворачивает фигуру по часовой стрелки на 90 градусов;
- `is_figure_out_of_bounds` – проверяет, вышла ли фигура за границы или столкнулась ли с другой фигурой;
- `insert_figure` – вписывает фигуру в поле — фигура становится частью поля;
- `move_figure left` – сдвигает фигуру влево;
- `move_figure right` – сдвигает фигуру вправо;
- `update_figures` – меняет активную фигуру на следующую фигуру, а следующую создает;
- `move_figure down` – сдвигает фигуру вниз;

Класс `view`:

- `place_game_over` – выводит на страницу экран конца игры (проигрыша);
- `place_top_scores` – возвращает таблицу рекордов;
- `place_panel` – выводит на страницу панель игры;
- `update_field` – очищает поле, выводит игру на страницу;
- `clear_field` – очищает поле;
- `place_cell` – рисует клетку/квадрат;

- `place_field` = выводит на страницу игровое поле;

Класс `controller`:

- `update_timer` – обновляет таймер игры и саму игру;
- `update_view` – заканчивает или обновляет состояние игры;
- `stop_timer` – останавливает таймер;
- `start_timer` – подключает таймер;
- `restart` – начинает игру заново;
- `key_up_uses` – выполнение события при отпуске клавиши;
- `key_down_uses` – выполнение события при нажатии клавиши;

Выводы.

Была изучена работа web-сервера `nginx` со статическими файлами и создано клиентское JavaScript web-приложение.

Приложение А. Исходный код.

Файл index.js

```
import Game from "../src/game.js";
import View from "../src/view.js";
import Controller from "../src/controller.js";

const element = document.querySelector('#root');//ссылка на root
(корневой элемент)
const game = new Game();
const view = new View(element, 1366, 768, 20, 10);
const controller = new Controller(game, view);

window.game = game; //из-за модулей константа (game) не попадает в
глобальное пространство имен
window.view = view;
window.controller = controller;
```

Файл game.js

```
export default class Game
{ //по умолчанию экспорт Game
  score = 0; //количество очков
  lines = 0; //количество "собранных" строк
  field = new Array(20); // игровое поле 20x10 (20 строк, 10 столбцов)
  figure_move = this.create_next_figure_move();
  next_figure_move = this.create_next_figure_move();
  is_end_top = false;
  best_scores = [0, 0, 0, 0, 0];

  make_best_scores()
  {
    this.best_scores.pop();
    this.best_scores.push(this.score);
    this.best_scores.sort(function (a, b){return b - a});
    localStorage.setItem("best_scores",
JSON.stringify(this.best_scores));
  }

  table_scores(best_scores)
  {
    if (localStorage.getItem("best_scores") === null ||
localStorage.getItem("best_scores") === undefined)
    {
      localStorage.setItem("best_scores",
JSON.stringify(best_scores));
    }
  }
}
```

```

        return JSON.parse(localStorage.getItem("best_scores"));
    }

    get level()
    {
        return Math.floor(this.lines * 0.1); // лвл, который меняется
        каждые 10 уровней (0-9 - 1, 10-19 - 2 и тд)
    }

    restart()
    {
        this.score = 0;
        this.lines = 0;
        this.is_end_top = false;
        this.field = this.create_field();
        this.figure_move = this.create_next_figure_move();
        this.next_figure_move = this.create_next_figure_move();
    }

    update_score(deleted_lines)
    {
        //console.log(deleted_lines);
        const points =
            {
                '1' : 40,
                '2' : 100,
                '3' : 300,
                '4' : 1200
            };
        if (deleted_lines > 0)
        {
            this.score += points[deleted_lines] * (this.level + 1);
            this.lines += deleted_lines;
            //console.log(this.score, this.lines);
        }
    }

    delete_lines()
    {
        const rows = 20;
        const columns = 10;
        let lines = []; //массив строк, которые нужно удалить

        for (let y = rows - 1; y >= 0; y--)
        {
            let blocks_count = 0; //количество непустых блоков на линии
(строке)
            for (let x = 0; x < columns; x++)

```



```

        {
            if (this.field[y][x] !== 0)
            {
                blocks_count++;
                //console.log(blocks_count);
            }
        }
        if (blocks_count === 0) //если линия пуста, то блоков сверху
нет
        {
            break;
        } else if (blocks_count < columns) //если линия не пуста, но
полностью не заполнена
        {
            continue;
        } else //оставшийся вариант - линия заполнена
        {
            lines.unshift(y); //добавим индекс строки в начало
        }
    }
    //console.log(lines);

    for (let i of lines)
    {
        this.field.splice(i, 1); // удаляем заполненную линию
        this.field.unshift(new Array(columns).fill(0)); //добавляем
пустую линию сверху поля
    }
    //console.log((lines.length));
    return lines.length;
}

create_next_figure_move()
{
    const figure_number = Math.floor(Math.random() * 7); //получение
числа от 0 до 7 (номер фигуры от 0 до 7)
    const figure_names = "IJLOSTZ"; //строка с именами фигур (1 буква
= 1 имя)
    const figure = figure_names[figure_number]; //получение имени
фигуры
    const figure_next = {};

    switch (figure)
    {
        case 'I':
            figure_next.blocks =
            [
                [0,0,0,0],

```

```

        [1,1,1,1],
        [0,0,0,0],
        [0,0,0,0],
    ];
    break;
case 'J':
    figure_next.blocks =
    [
        [0,0,0],
        [2,2,2],
        [0,0,2]
    ];
    break;
case 'L':
    figure_next.blocks =
    [
        [0,0,0],
        [3,3,3],
        [3,0,0]
    ];
    break;
case 'O':
    figure_next.blocks =
    [
        [0,0,0,0],
        [0,4,4,0],
        [0,4,4,0],
        [0,0,0,0]
    ];
    break;
case 'S':
    figure_next.blocks =
    [
        [0,0,0],
        [0,5,5],
        [5,5,0]
    ];
    break;
case 'T':
    figure_next.blocks =
    [
        [0,0,0],
        [6,6,6],
        [0,6,0]
    ];
    break;
case 'Z':
    figure_next.blocks =

```

```

        [
            [0,0,0],
            [7,7,0],
            [0,7,7]
        ];
        break;
    default:
        throw new Error("Неизвестная фигура!");
    }

    figure_next.x = Math.floor((10 -
figure_next.blocks[0].length)/2); //верхний центр (чтобы фигура падала с
середины)
    figure_next.y = -1;
    return figure_next;
}

get_field()
{
    const field = this.create_field(); //копия поля
    for (let y = 0; y < this.field.length; y++)
    {
        for (let x = 0; x < this.field[y].length; x++)
        {
            field[y][x] = this.field[y][x];
        }
    }

    for (let y = 0; y < this.figure_move.blocks.length; y++) //копия
падающей фигуры на поле
    {
        for (let x = 0; x < this.figure_move.blocks[y].length; x++)
        {
            if (this.figure_move.blocks[y][x] !== 0)
            {
                field[this.figure_move.y + y][this.figure_move.x + x]
= this.figure_move.blocks[y][x]; //координата+отступ
            }
        }
    }
    return{
        next_figure_move : this.next_figure_move,
        score : this.score,
        level: this.level,
        lines: this.lines,
        field,
        is_game_over : this.is_end_top
    };
};

```

```

}

create_field()
{
    const field = new Array(this.field.length)
    for (let y = 0; y < field.length; y++)
    {
        field[y] = new Array(this.field[y].length);
        for (let x = 0; x < field[y].length; x++)
        {
            field[y][x] = 0;
        }
    }
    return field;
}

rotate_figure_left()
{
    this.rotate_figure();
    this.rotate_figure();
    this.rotate_figure();
}

rotate_figure() //поворот падающей фигуры на 90 по часовой
{
    const blocks = this.figure_move.blocks; //на случай выхода за
    границы или столкновений с другими фигурами
    const temp = new Array(this.figure_move.blocks.length); //массив
    для поворота фигуры
    for (let i = 0; i < temp.length; i++)
    {
        temp[i] = new Array(temp.length);
        for (let j = 0; j < temp[i].length; j++)
        {
            temp[i][j] = 0;
        }
    }

    for (let y = 0; y < temp.length; y++)
    {
        for (let x = 0; x < temp[y].length; x++)
        {
            temp[x][y] = this.figure_move.blocks[temp.length - 1 - y]
            [x]; //замена строк на столбцы (поворот)
        }
    }
    this.figure_move.blocks = temp; //замена падающей фигуры на
    повернутую
}

```

```

        if (this.is_figure_out_of_bounds()) //в случае выхода за пределы
        {
            this.figure_move.blocks = blocks;
        }
    }

    is_figure_out_of_bounds()
    {
        for (let y = 0; y < this.figure_move.blocks.length; y++)
        {
            for (let x = 0; x < this.figure_move.blocks[y].length; x++)
            {
                if (this.figure_move.blocks[y][x] !== 0 && //если в поле
фигуры 0, то все равно на пределы поля игры и другие фигуры
                (this.field[this.figure_move.y + y] === undefined ||
                this.field[this.figure_move.y + y][this.figure_move.x
+ x] === undefined ||
                this.field[this.figure_move.y + y][this.figure_move.x
+ x] !== 0))
                {
                    return true;
                }
            }
        }
        return false;
    }

    insert_figure()
    {
        for (let y = 0; y < this.figure_move.blocks.length; y++)
        {
            for (let x = 0; x < this.figure_move.blocks[y].length; x++)
            {
                if (this.figure_move.blocks[y][x] !== 0) //нули фигуры не
должны переписываться
                {
                    this.field[this.figure_move.y + y][this.figure_move.x
+ x] = this.figure_move.blocks[y][x];
                }
            }
        }
    }

    move_figure_left()
    {
        this.figure_move.x -= 1;
        if (this.is_figure_out_of_bounds())

```

```

        {
            this.figure_move.x += 1;
        }
    }

    move_figure_right()
    {
        this.figure_move.x += 1;
        if (this.is_figure_out_of_bounds())
        {
            this.figure_move.x -= 1;
        }
    }

    update_figures() //меняем падающую фигуру на следующую
    {
        this.figure_move = this.next_figure_move;
        this.next_figure_move = this.create_next_figure_move();
    }

    move_figure_down()
    {
        if (this.is_end_top)
        {
            return;
        }

        if (this.figure_move.y + 1 < 20)
        {
            this.figure_move.y += 1;
            if (this.is_figure_out_of_bounds())
            {
                this.figure_move.y -= 1;
                this.insert_figure();
                const deleted_lines = this.delete_lines();
                this.update_score(deleted_lines);
                this.update_figures();
            }
        }

        if (this.is_figure_out_of_bounds())
        {
            this.is_end_top = true;
        }
    }

    constructor()
    {

```

```

    for (let i = 0; i < this.field.length; i++)
    {
        this.field[i] = new Array(10);
        for (let j = 0; j < this.field[0].length; j++)
        {
            this.field[i][j] = 0;
        }
    }
}

```

файл view.js

```

export default class View
{
    constructor(element, width, height, rows, columns)
    {
        this.element = element;
        this.width = width;
        this.height = height;
        this.canvas = document.createElement('canvas'); //создание холста
        this.canvas.width = this.width;
        this.canvas.height = this.height;
        this.ctx = this.canvas.getContext('2d'); //доступ к 2д фигурам и
        графике
        //this.name

        this.field_border_width = 4; //ширина границы игрового поля
        this.field_x = this.field_border_width; //координаты верхнего
        левого угла поля
        this.field_y = this.field_border_width;
        this.field_width = this.width * 1/2; // 2/3 поля будет 2/3 от
        общей
        this.field_height = this.height;
        this.field_inner_width = this.field_width -
        this.field_border_width*2; //внутренняя часть поля
        this.field_inner_height = this.field_height -
        this.field_border_width;

        //this.rows = rows;
        //this.columns = columns;

        this.cell_width = this.field_inner_width / columns; //ширина
        одной клетки на поле (в пикселях)
        this.cell_height = this.field_inner_height / rows; //высота одной
        клетки на поле (в пикселях)

        this.panel_x = this.field_width + 10; //координаты панели

```

```

        this.panel_y = 0;
        //this.panel_width = this.width * 1/2;
        //this.panel_height = this.height;

        this.element.appendChild(this.canvas); //добавление контекста в
передаваемый элемент
    }

    place_game_over({score})
    {
        this.clear_field();
        this.ctx.fillStyle = 'black'; //цвет текста
        this.ctx.font = '14px "Crimson+Pro"';
        this.ctx.textAlign = 'center'; //по центру
        this.ctx.textBaseline = 'middle';
        this.ctx.fillText('GAME OVER', this.width/2+75, this.height/2 -
48);
        this.ctx.fillText(`Score: ${score}`, this.width/2+75,
this.height/2);
        this.ctx.fillText("Press ENTER to restart", this.width/2+75,
this.height/2 + 48);

        this.ctx.fillText(`Top scores:`, this.width/2+75, this.height/2 +
48+24);

        this.ctx.textAlign = 'left';
        this.ctx.textBaseLine = 'bottom';
        let best_scores = this.place_top_scores();
        for (let i = 0; i < best_scores.length; i++)
        {
            this.ctx.fillText(`${i+1}) ${best_scores[i]}`, this.width/2-
20+75, this.height/2 +48+24*(2+i));
        }
    }

    place_top_scores()
    {
        let best_scores =
JSON.parse(localStorage.getItem("best_scores"));

        if (best_scores === null || best_scores === undefined)
        {
            best_scores = [0, 0, 0, 0, 0];
        }

        return best_scores;
    }
}

```



```

place_panel({next_figure_move, score, level})
{
    let colors =
        {
            '1' : 'red',
            '2' : 'green',
            '3' : 'blue',
            '4' : 'pink',
            '5' : 'yellow',
            '6' : 'purple',
            '7' : 'orange'
        };

    const col_2 =
        {
            '1' : 'HotPink',
            '2' : 'Crimson',
            '3' : 'DarkBlue',
            '4' : 'DarkOrange',
            '5' : 'DeepPink'
        }

    this.ctx.textAlign = 'left'; // справа
    this.ctx.textBaseline = 'top'; // по верхнему краю
    this.ctx.fillStyle = 'black'; //цвет текста
    this.ctx.font = '14px "Crimson+Pro"'; //шрифт
    //this.ctx.fillStyle = 'red';
    this.ctx.fillText(`Level: ${level}`, this.panel_x, this.panel_y);
    this.ctx.fillText(`Score: ${score}`, this.panel_x, this.panel_y +
24);

    let name = localStorage.getItem('#tetris.name_input');
    this.ctx.fillText(`Your name: ${name}`, this.panel_x,
this.panel_y + 48);

    //let scores = localStorage.getItem('best_scores');

    //this.place_top_scores(this.panel_x, this.panel_y + 72);
    let best_scores = this.place_top_scores();
    this.ctx.fillText(`Top scores:`, this.panel_x, this.panel_y +
48+24);
    for (let i = 0; i < best_scores.length; i++)
    {
        this.ctx.fillStyle = col_2[Math.floor(Math.random() * 5)];
        this.ctx.fillText(`${i+1}) ${best_scores[i]}`, this.panel_x,
this.panel_y + 48+24*(2+i));
    }
}

```

```

        //this.ctx.fillText(`Top scores: $
{JSON.parse(localStorage.getItem("best_scores"))}`,
        //this.panel_x, this.panel_y + 48 + 24);

        this.ctx.fillText("Next figure:", this.panel_x, this.panel_y + 48
+ 24*7);

        for (let y = 0; y < next_figure_move.blocks.length; y++)
//следующая фигура
        {
            for (let x = 0; x < next_figure_move.blocks[y].length; x++)
            {
                if (next_figure_move.blocks[y][x] !== 0)
                {
                    this.place_cell(x*this.cell_width/2 + this.panel_x,
y*this.cell_height/2 + this.panel_y + 225,
                    this.cell_width/2, this.cell_height/2,
colors[next_figure_move.blocks[y][x]]);
                }
            }
        }

        this.ctx.fillText("Control:", this.panel_x, this.panel_y + 300);
        this.ctx.fillText("Left arrow - Left move", this.panel_x,
this.panel_y + 324);
        this.ctx.fillText("Right arrow - Right move", this.panel_x,
this.panel_y + 348);
        this.ctx.fillText("Down arrow - Falling down", this.panel_x,
this.panel_y + 372);
        this.ctx.fillText("'Shift\' - Counterclockwise rotation",
this.panel_x, this.panel_y + 396);
        this.ctx.fillText("Up arrow - Clockwise rotation", this.panel_x,
this.panel_y + 420);

    }

    update_field(condition)
    {
        this.clear_field();
        this.place_field(condition);
        this.place_panel(condition);
    }

    clear_field()
    {
        this.ctx.clearRect(0,0, this.width, this.height);
    }

```

```

place_cell(x, y, width, height, color)
{
    //console.log({y, x});
    this.ctx.fillStyle = color; //заливка
    this.ctx.strokeStyle = 'black'; //обводка
    this.ctx.lineWidth = 2; //ширина обводки
    this.ctx.fillRect(x, y, width, height);
    this.ctx.strokeRect(x, y, width, height); // обводка
    //размещение фигуры (по клеточно в пикселях с шириной одной клетки
на поле)
}

place_field({field})
{
    let colors =
    {
        '1' : 'red',
        '2' : 'green',
        '3' : 'blue',
        '4' : 'pink',
        '5' : 'yellow',
        '6' : 'purple',
        '7' : 'orange'
    };
    //console.log(field);
    for (let y = 0; y < field.length; y++)
    {
        for (let x = 0; x < field[y].length; x++)
        {
            if (field[y][x] !== 0)
            {
                this.place_cell(x*this.cell_width + this.field_x,
y*this.cell_height + this.field_y,
                this.cell_width, this.cell_height,
colors[field[y][x]])
            }
        }
    }

    this.ctx.strokeStyle = "black"; //обводка
    this.ctx.lineWidth = this.field_border_width;
    this.ctx.strokeRect(this.field_border_width,
this.field_border_width,
        this.field_width-this.field_border_width-1,
this.field_height-this.field_border_width);
    }
}

```

файл controller.js

```
export default class Controller
{
  constructor(game, view)
  {
    this.count = 0;
    this.game = game;
    this.view = view;
    this.interval_id = null;

    this.start_timer();

    document.addEventListener('keydown',
this.key_down_uses.bind(this)); //регистрируем событие при нажатии на
клавишу
    document.addEventListener('keyup',
this.key_up_uses.bind(this));

    view.update_field(game.get_field());
  }

  update_timer()
  {
    //localStorage.setItem("best_scores",
JSON.stringify(game.best_scores));
    this.stop_timer();
    this.game.move_figure_down();
    this.start_timer();
    this.update_view();
    //this.view.update_field(this.game.get_field());
  }

  update_view()
  {
    const state = this.game.get_field();

    if (state.is_game_over)
    {
      this.count++;
      if (this.count === 1)
      {
        if (this.game.score > this.game.best_scores[4])
        {
          this.game.best_scores =
```

```

this.game.table_scores(this.game.best_scores);
        this.game.make_best_scores();
        this.game.best_scores =
this.game.table_scores(this.game.best_scores);
    }
    }
    this.view.place_game_over(state);
}
else
{
    this.view.update_field(state);
}
}

stop_timer()
{
    if (this.interval_id)
    {
        clearInterval(this.interval_id);
        this.interval_id = null;
    }
}

start_timer()
{
    const speed = 1000 - this.game.get_field().level * 100;
//скорость падения

    if (!this.interval_id)
    {
        this.interval_id = setInterval(() => {
            this.update_timer();
        }, speed > 0? speed: 100);
    }
}

restart()
{/*
    if (this.game.score > this.game.best_scores[4])
    {
        this.game.best_scores =
this.game.table_scores(this.game.best_scores);
        this.game.make_best_scores();
        this.game.best_scores =
this.game.table_scores(this.game.best_scores);
    }*/
    this.game.restart();
    this.start_timer();

```

```

        this.update_view();
    }

    key_up_uses(event)
    {
        switch (event.keyCode)
        {
            case 40: //стрелка вниз
                this.start_timer();
                break;
        }
    }

    key_down_uses(event)
    {
        const state = this.game.get_field();

        switch (event.keyCode) //keyCode возвращает числовой код
КЛАВИШИ
        {
            case 37: //НАЛЕВО(стрелка)
                this.game.move_figure_left();
                this.update_view();
                break;
            case 38: //ВВЕРХ(стрелка)
                this.game.rotate_figure();
                this.update_view();
                break;
            case 39: //ВПРАВО(стрелка)
                this.game.move_figure_right();
                this.update_view();
                break;
            case 40: //ВНИЗ(стрелка)
                this.stop_timer();
                this.game.move_figure_down();
                this.update_view();
                break;
            case 16: //SHIFT
                this.game.rotate_figure_left();
                this.update_view();
                break;
            case 13: //ENTER
                if (state.is_game_over)
                {
                    this.restart();
                }
        }
    }

```

```
    }  
}
```

файл index.html

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <title>Tetris</title>  
  <style type="text/css">  
    body{  
      background: url(1.webp);  
    }  
  </style>  
</head>  
<body onload="read()">  
  <h1 align="middle">Tetris</h1>  
  <form action = "main.html" method = "get" id="name_input_f"  
align="middle">  
    <div>  
      <label style = "color:white">  
        Enter your name:  
      </label></br>  
      <input style="color:blue" placeholder="Enter your name"  
id="name_input">  
    </div>  
    <button type="submit" onclick="store()">Submit</button>  
    <script>  
      function store() {  
        if (document.getElementById('name_input').value === null)  
{  
          alert("Enter the name!");  
        } else  
          localStorage.setItem('#tetris.name_input',  
document.getElementById('name_input').value);  
      }  
  
      function read() {  
        if  
(localStorage.hasOwnProperty("#tetris.name_input"))  
        {  
          let name =  
localStorage.getItem("#tetris.name_input");  
          let input =  
document.getElementById("name_input");  
          input.value = name;  
        }  
      }  
    </script>  
  </form>  
</body>  
</html>
```

```

        return name
    }
}
</script>
</form>
</body>
</html>

файл main.html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Tetris</title>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Crimson+Pro">
    <style type="text/css">
        body{
            background: url(2.webp);
        }
    </style>
</head>
<body>
    <div id="root">
        <div align="right">
            <div style="position: absolute; top: 80%; left: 48%">
                <form action="index.html" method="get">
                    <button type="submit">Go home</button>
                </form>
            </div>
        </div>
    </div>

    <script src = "../index.js" type = "module"></script>
    <script src = "../src/game.js" type = "module"></script>
    <script src = "../src/view.js" type = "module"></script>
    <script src = "../src/controller.js" type = "module"></script>

</body>
</html>

```