**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра математического обеспечения и применения ЭВМ**

# ОТЧЕТ

**По лабораторной работе № 1**
**по дисциплине «Web-Технологии»**
**Тема: Тетрис на JavaScript**

Студент гр. 0303 _____ Болкунов В.О.

Преподаватель _____ Беляев С. А.

Санкт-Петербург
2022

**Цель работы.**

Целью работы является изучение работы web-сервера nginx со статическими файлами и создание клиентских JavaScript web-приложений. Для достижения поставленной цели требуется решить следующие задачи: – генерация открытого и закрытого ключей для использования шифрования (https://www.openssl.org/); – настройка сервера nginx для работы по протоколу HTTPS; – разработка интерфейса web-приложения; – обеспечение ввода имени пользователя; – обеспечение создания новой фигуры для тетриса по таймеру и ее движение; – обеспечение управления пользователем падающей фигурой; – обеспечение исчезновения ряда, если он заполнен; – по окончании игры – отображение таблицы рекордов, которая хранится в браузере пользователя.

**Общая формулировка задачи.**

Необходимо создать web-приложение – игру в тетрис. Основные требования: – сервер – nginx, протокол взаимодействия – HTTPS версии не ниже 2.0;

– отображается страница для ввода имени пользователя;

– статическая страница отображает «стакан» для тетриса, следующей фигуры, и имени пользователя;

– фигуры в игре – классические фигуры тетриса (7 шт. тетрамино); – случайным образом генерируется фигура и начинает падать в «стакан» (описание правил см., например, https://ru.wikipedia.org/wiki/Тетрис);

– пользователь имеет возможность двигать фигуру влево и вправо, повернуть на 90° и «уронить»;

– если собралась целая «строка», она должна исчезнуть;

– при наборе некоторого заданного числа очков увеличивается уровень, что заключается в увеличении скорости игры;

– пользователь проигрывает, когда стакан «заполняется», после чего ему отображается локальная таблица рекордов;

**Выполнение работы.**

**Модель и логика игры.**

Модель игры описывается следующими структурами данных.

- Вспомогательные типы-синонимы для кортежа двух чисел и матрицы.

```
type Num2Tuple = [number, number];
type Matrix<T> = Array<Array<T>>;
```

- Вспомогательные перечисления для движения игрока и для результата движения.

```
enum Actions {
    LEFT, ROTATE, RIGHT, DOWN
}
enum MoveResult {
        CHANGE,       - были изменения в модели (движение фигуры)
        CHANGELESS,- изменений не было (например фигура упёрлась)
        PLACE,        - фигура установлена на поле
        EMPTY         - активная фигура отсутствует
}
```

- Двумерный вектор, содержит информацию о положении клетки на поле, также используется для задания центра фигуры (который может не лежать на конкретной клетке). Содержит методы для векторных вычислений (translate, rotate, neg, round) и статические фабричные методы стандартных векторов 2-ух-мерного пространства.

```
class Vec2
    x: number;
    y: number;
    constructor(vec: Num2Tuple)
    translate(v: Vec2): Vec2
```

```
    neg(): Vec2
    rotate(mat: Matrix<number>): Vec2
    round(): Vec2
    tuple(): Num2Tuple
    static null(): Vec2
    static up(): Vec2
    static down(): Vec2
    static left(): Vec2
    static right(): Vec2
```

- Клетка на поле, либо в фигуре, содержит положение (вектор) и свой цвет.

```
class Cell
    pos: Vec2;
    color: string;
    constructor(pos: Vec2, color: string = '#000')
```

- Фигура (текущая фигура управляемая игроком). Содержит масив своих клеток, координату центра, относительно которого совершается поворот, цвет и методы управления положением (аналогично вектору). Методы min/max возвращают максимальные/минимальные координаты клеток фигуры по разным осям, это требуется для выравнивая фигуры после поворота около стенки стакана.

```
class Figure
    cells: Cell[];
    center: Vec2;
    color: string;
    constructor(cells: Array<Num2Tuple>, center: Num2Tuple, color: string)
    max(): Vec2
    min(): Vec2
```

```
    translate(v: Vec2)
    rotate(mat: Matrix<number> = rotMat90)
    copy()
```

- Игровое поле. Содержит матрицу клеток (могут быть пустыми), и методы для управления полем: поставить фигуру, очистить ряд, или уронить ряд. Метод canPlace используется для проверки возможности установки фигуры с учётом её сдвига.

```
class Field
    height: number;
    width: number;
    mat: Matrix<Cell | null>;
    constructor(h: number, w: number)
    canPlace(fig: Figure, move: Vec2 = Vec2.null()): boolean
    place(fig: Figure)
    checkRows(figure: Figure): number[]
    clearRow(row: number)
    fallDownRow(i: number)
```

- Класс игры, реализует логику основных действий.

Содержит игровое поле, ссылку на активную фигуру и очередь фигур. Предоставляет интерфейс управления игрой – метод move, методы помещения фигуры в очередь и добавления фигуры из очереди на поле, метод очистки занятых рядов и метод для подсказки места падения фигуры.

```
class Game
    field: Field;
    activeFigure: Figure | null;
    figuresQueue: Array<Figure> = [];
    filledRows: number[] = [];
```

```
constructor(h: number, w: number)
pushFigure(fig: Figure)
shiftFigure(): boolean
private clearRow(row: number)
clearFilledRows(): number
private moveDown(): MoveResult
private moveLR(act: Actions.LEFT | Actions.RIGHT): MoveResult
private rotate(): MoveResult
move(act: Actions): MoveResult
getHint(): Figure | null
```

**Интерфейс пользователя**.

Пользовательский интерфейс сделан с использованием компонентов интерфейса фреймворка *vue* и библиотек *vue-class-component* и *vue-property-decorator*.

- Вспомогательные интерфейсы/перечисления

Интерфейс передачи состояния из компоненты модели в компонент отображения.

```
interface GameViewState {
  readonly field?: Field;
  readonly figure?: Figure | null;
  readonly hint?: Figure | null;
}
```

Перечисление звуковых событий.

```
enum SoundEvents {
  MOVE, PLACE, CLEAR, OVER
}
```

Интерфейс объекта хранения рекордов.

```
interface Records {
  records: Array<[string, number]>;
}
```

- TetrisModel является гибко конфигурируемой прослойкой между логикой действий игры и компонентами отображения. Предоставляет интерфейс для команд пользователя, реализует таймер авто-сдвига вниз и генерирует события изменения отображений, очков, окончания игры и звуковые события. Все эти события могут быть обработаны родительским компонентом.

```
@Component
class TetrisModel extends Vue
  @Prop() width!: number;
  @Prop() height!: number;
  @Prop({default: 1500}) defaultInterval!: number;
  @Prop({default: 0.3}) intervalScoreMultiplier!: number;
  @Prop({default: 100}) rowScore!: number;
  private game!: Game;
  private timerId: number = 0;
  private score: number = 0;
  private paused: boolean = true;
  created()
  getInterval(): number
  private generateFigure()
  private spawnFigure(): boolean
  private tick()
  pause()
  resume()
  move(act: Actions)
```

```
  @Emit('model-change')
  private modelChangeEmit(): GameViewState
  @Emit('next-change')
  private nextChangeEmit(): GameViewState
  @Emit('score-change')
  private scoreChangeEmit(): number
  @Emit('game-over')
  private gameOverEmit(): boolean
  @Emit('sound-event')
  private soundEmit(sound: SoundEvents): SoundEvents
```

- TetrisView – универсальный компонент отображения, может быть использован как для отображения самой игры, так и отдельных фигур (например следующей).

```
@Component
export default class TetrisView extends Vue
  @Prop() width!: number;
  @Prop() height!: number;
  @Prop({default: 20}) cellSize!: number;
  @Prop({default: 1}) borderSize!: number;
  @Prop({default: '#fff'}) color!: string;
  @Prop({default: '#fff'}) bgColor!: string;
  @Prop({default: '#777'}) borderColor!: string;
  $refs!: {
    field: HTMLDivElement
    rows: HTMLDivElement[]
    cells: HTMLDivElement[]
  }
  update(state: GameViewState)
```

- Tetris – цельный компонент игры, агрегирует модель, и отображения, отлавливает нажатия клавиш и проигрывает звуковые события, также оповещает о поражении.

```
@Component
class Tetris extends Vue
  width: number = 10;
  height: number = 20;
  cellSize: number = 30;
  sounds: string[] =
     ['move', 'place', 'clear', 'game_over']
        .map(name => require(`@/sounds/${name}.wav`));
  score: number = 0;
  over: boolean = false;
  playerName: string = localStorage.getItem('tetris.name') ?? '';
  $refs!: {
   model: TetrisModel,
   view: TetrisView,
   next: TetrisView,
  }
  private onKeyDown(e: KeyboardEvent)
  private onResize(e?: UIEvent)
  mounted()
  beforeDestroy()
  playSound(event: SoundEvents)
  gameOver()
```

**Компоненты представлений роутера**

Используются для навигации по приложению.

```
@Component
export default class GameView extends Vue
```

Компонент HomeView используется для ввода имени пользователя.

```
@Component
class HomeView extends Vue
  name: string = '';
  $refs!: {
    input: HTMLInputElement;
  }
  mounted()
  start()
  focus()
```

Компонент RecordsView используется для отображения таблицы рекордов.

```
@Component
export default class RecordsView extends Vue
  records!: Records;
  created()
```

**Примеры экранных форм**.

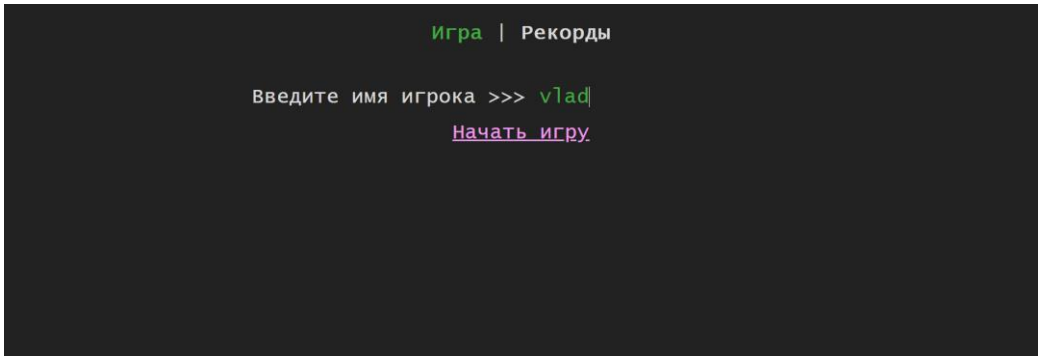


Рисунок 1: Стартовая страница


Рисунок 2: Игровой процесс

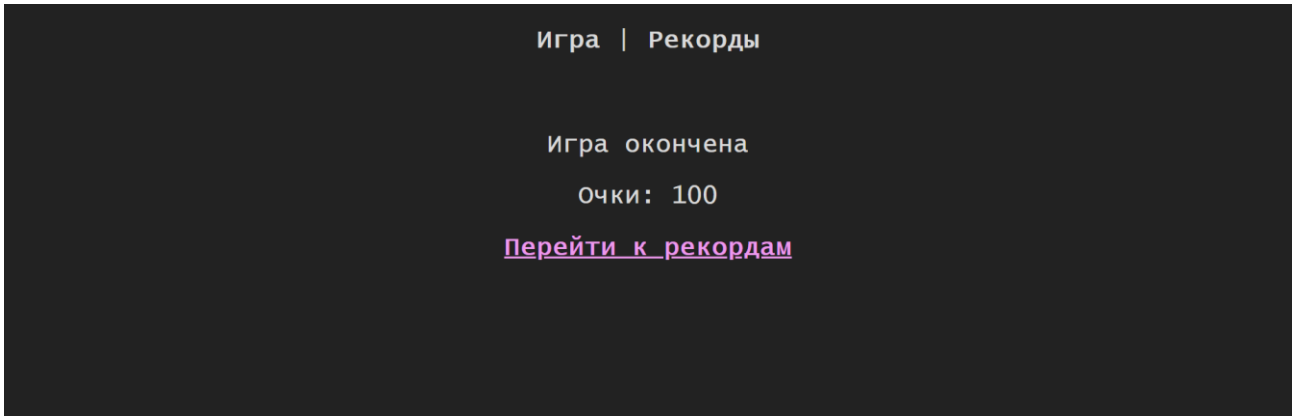


Рисунок 3: Элемент оповещения о поражении

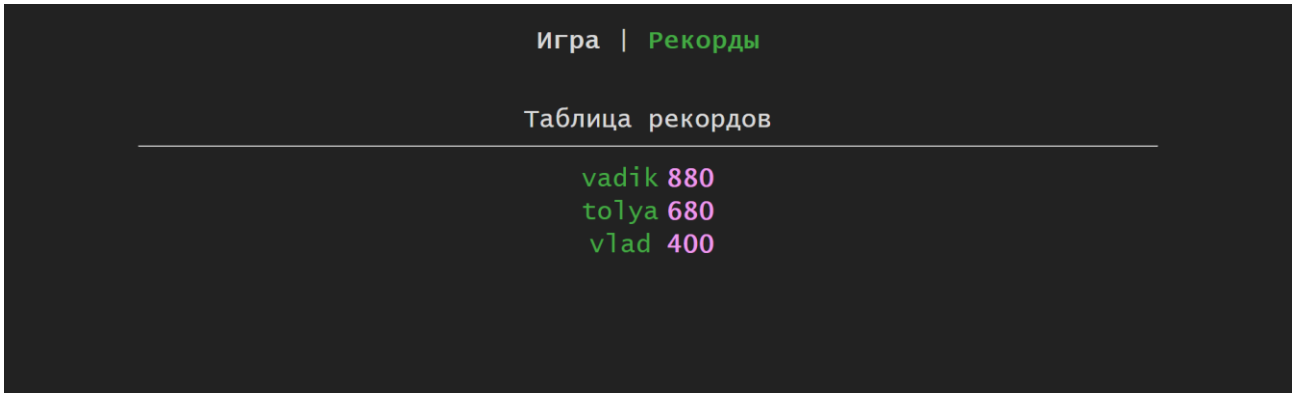


Рисунок 4: Таблица рекордов

**Настройка web-сервера nginx.**

С помощью утилиты openssl были сгенерированы приватный ключ и сертификат, обеспечивающие шифрованное соединений. Для сервера nginx была применена следующая конфигурация:

```
server {
  listen 80;
  server_name localhost;
  return 301 https://localhost;
}

server {
  listen 443 ssl http2;
  server_name localhost;
  ssl_certificate /home/vlad/ssl/cert.crt;
  ssl_certificate_key /home/vlad/ssl/key.key;
  charset utf-8;
  root /home/vlad/WebstormProjects/WebLabs/dist;
  index index.html;
}
```

Данная конфигурация обеспечивает переадресацию с http соединения на https.

Также для проверки работоспособности web-приложения и конфигурации веб-сервера был настроен сервер в сети интернет доступный по адресу https://game.wex-web.ru/.

**Выводы.**

В ходе работы были изучены инструменты работы с web-сервером nginx, способного предоставлять доступ к клиентским веб-приложениям. Сгенерированы ключи для работы с по защищённому протоколу https, установлена конфигурация nginx и реализовано web приложение «Тетрис» с помощью языка TypeScript и фреймворка Vue2.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

## *Файл tetris/util.ts*

```
import {Figure} from "./Figure";

// Кортеж двух чисел
export type Num2Tuple = [number, number];

// Шаблон матрицы
export type Matrix<T> = Array<Array<T>>;

// Матрица поворота на 90
export const rotMat90: Matrix<number> = [
    [0, -1],
    [1, 0]
];

// Возможные действия с фигурой
export enum Actions {
    LEFT, ROTATE, RIGHT, DOWN
}

// Фабричные функции стандартных фигур
export const FiguresFabric = [
    () => new Figure([[0, 0], [1, 0], [2, 0], [3, 0]], [1.5, 0],
'#729fcf'),
    () => new Figure([[0, 0], [1, 0], [0, 1], [1, 1]], [0.5, 0.5],
'#d3d7cf'),
    () => new Figure([[0, 0], [0, 1], [0, 2], [1, 2]], [0.5, 1.5],
'#c4a000'),
    () => new Figure([[1, 0], [1, 1], [1, 2], [0, 2]], [0.5, 1.5],
'#ee99ee'),
    () => new Figure([[0, 0], [0, 1], [1, 1], [1, 2]], [0.5, 1],
'#cc0000'),
    () => new Figure([[1, 0], [1, 1], [0, 1], [0, 2]], [0.5, 1],
'#4e9a06'),
    () => new Figure([[0, 0], [0, 1], [0, 2], [1, 1]], [1, 1],
'#75507b'),
]
```

## *Файл tetris/Vec2.ts*

```
import {Matrix, Num2Tuple} from "./util";

// Двумерный вектор
export class Vec2 {
    x: number;
    y: number;

    constructor(vec: Num2Tuple) {
        this.x = vec[0];
        this.y = vec[1];
    }

    // Сумма векторов
```

```
    translate(v: Vec2): Vec2 {
        return new Vec2([this.x + v.x, this.y + v.y]);
    }

    // Разворот вектора
    neg(): Vec2 {
        return new Vec2([-this.x, -this.y]);
    }

    // Поворот вектора
    rotate(mat: Matrix<number>): Vec2 {
        return new Vec2([
            this.x * mat[0][0] + this.y * mat[0][1],
            this.x * mat[1][0] + this.y * mat[1][1],
        ]);
    }

    // Округление координат
    round(): Vec2 {
        return new Vec2([Math.round(this.x), Math.round(this.y)]);
    }

    // Преобразование к кортежу
    tuple(): Num2Tuple {
        return [this.x, this.y];
    }

    // Фабричные методы стандартных векторов
    static null(): Vec2 {
        return new Vec2([0, 0]);
    }

    static up(): Vec2 {
        return new Vec2([0, -1]);
    }

    static down(): Vec2 {
        return new Vec2([0, 1]);
    }

    static left(): Vec2 {
        return new Vec2([-1, 0]);
    }

    static right(): Vec2 {
        return new Vec2([1, 0]);
    }
}
```

### *Файл tetris/Cell.ts*

```
import {Vec2} from "./Vec2";

// Клетка на поле (или в фигуре)
export class Cell {
    pos: Vec2;
    color: string;

    constructor(pos: Vec2, color: string = '#000') {
        this.pos = pos;
        this.color = color;
```

```
    }
}
```

### *Файл tetris/Figure.ts*

```
import {Matrix, rotMat90, Num2Tuple} from "./util";
import {Vec2} from "./Vec2";
import {Cell} from "./Cell";

// Фигура на поле (текущая управляемая фигура)
export class Figure {
    cells: Cell[];
    // Центр фигуры задаёт ось её вращения
    center: Vec2;
    color: string;

    constructor(cells: Array<Num2Tuple>, center: Num2Tuple, color: string) {
        this.cells = cells.map(v => new Cell(new Vec2(v), color));
        this.center = new Vec2(center);
        this.color = color;
    }

    // Максимальные x и y координаты фигуры
    max(): Vec2 {
        return new Vec2([
            Math.max(...this.cells.map(v => v.pos.x)),
            Math.max(...this.cells.map(v => v.pos.y))
        ])
    }

    // Минимальные x и y координаты фигуры
    min(): Vec2 {
        return new Vec2([
            Math.min(...this.cells.map(v => v.pos.x)),
            Math.min(...this.cells.map(v => v.pos.y))
        ])
    }

    // Перемещение фигуры целиком
    translate(v: Vec2) {
        this.center = this.center.translate(v);
        for (let i = 0; i < this.cells.length; i++) {
            this.cells[i].pos = this.cells[i].pos.translate(v);
        }
    }

    // Поворот фигуры
    rotate(mat: Matrix<number> = rotMat90) {
        this.cells.forEach(cell =>
            cell.pos =
                cell.pos
                    .translate(this.center.neg())
                    .rotate(mat)
                    .translate(this.center)
                    .round()
        );
    }

    // Копия фигуры
    copy() {
        return new Figure(
            this.cells.map(cell => cell.pos.tuple()),
```

```
            this.center.tuple(),
            this.color
        )
    }
}
```

### *Файл tetris/Field.ts*

```
import {Matrix} from "./util";
import {Vec2} from "./Vec2";
import {Cell} from "./Cell";
import {Figure} from "./Figure";

const _ = require('lodash');

export class Field {
    height: number;
    width: number;
    mat: Matrix<Cell | null>;

    constructor(h: number, w: number) {
        this.height = h;
        this.width = w;
        this.mat = Array(h);
        for (let i = 0; i < h; i++) {
            this.mat[i] = _.fill(Array(w), null);
        }
    }

    // Определение возможности размещения фигуры (с учётом её сдвига)
    canPlace(fig: Figure, move: Vec2 = Vec2.null()): boolean {
        for (let c of fig.cells) {
            if (
                c.pos.x + move.x < 0 ||
                c.pos.y + move.y < 0 ||
                c.pos.x + move.x >= this.width ||
                c.pos.y + move.y >= this.height ||
                this.mat[c.pos.y + move.y][c.pos.x + move.x] != null
            )
                return false;
        }
        return true;
    }

    // Помещение фигуры (её клеток) на поле
    place(fig: Figure) {
        if (!this.canPlace(fig))
            throw new Error(`Incorrect/occupied position; figure:
${JSON.stringify(fig)}`);
        for (let c of fig.cells)
            this.mat[c.pos.y][c.pos.x] = c;
    }

    // Получение заполненных фигурой рядов
    checkRows(figure: Figure): number[] {
        return _.uniq(figure.cells.map(c => c.pos.y))
            .filter(
                (y: number) =>
                    this.mat[y].every(
                        cell => cell != null
                    )
            );
```

```
    }

    // Очищение ряда клеток
    clearRow(row: number) {
        for (let col = 0; col < this.width; col++)
            this.mat[row][col] = null;
    }

    // Падение ряда на следующий ряд
    fallDownRow(i: number) {
        for (let j = 0; j < this.width; j++) {
            if (this.mat[i + 1][j] == null && this.mat[i][j] != null) {
                this.mat[i][j]!.pos.y += 1;
                this.mat[i + 1][j] = this.mat[i][j];
                this.mat[i][j] = null;
            }
        }
    }
}
```

### *Файл tetris/Game.ts*

```
import {Actions, rotMat90} from "./util";
import {Vec2} from "./Vec2";
import {Figure} from "./Figure";
import {Field} from "./Field";

// Результат перемещения фигуры
export enum MoveResult {
    CHANGE, CHANGELESS, PLACE, EMPTY
}

export class Game {
    // Поле клеток
    field: Field;
    // Управляемая фигура
    activeFigure: Figure | null;
    figuresQueue: Array<Figure> = [];
    filledRows: number[] = [];

    constructor(h: number, w: number) {
        this.field = new Field(h, w);
        this.activeFigure = null;
    }

    // Помещение фигуры в очередь
    pushFigure(fig: Figure) {
        this.figuresQueue.push(fig);
    }

    // Спавн фигуры из очереди, возвращает true, если фигура размещена удачно
(false - поражение)
    shiftFigure(): boolean {
        if (!this.figuresQueue.length)
            throw new Error('Empty figures queue');

        let fig = this.figuresQueue.shift();
        fig!.translate(
            new Vec2([
                this.field.width / 2 - fig!.center.x - 1, 0
            ]).round()
        );
```

```
        if (this.field.canPlace(fig!)) {
            this.activeFigure = fig!;
            return true;
        } else
            return false;
    }

    // Очистка одного ряда и сдвиг вышестоящих вниз
    private clearRow(row: number) {
        this.field.clearRow(row);
        for (let i = row - 1; i >= 0; --i) {
            this.field.fallDownRow(i);
        }
    }

    // Очистка всех заполненных рядов
    clearFilledRows(): number {
        let rows = this.filledRows.length;
        this.filledRows
            .sort((a, b) => a - b)
            .forEach(row => this.clearRow(row));
        this.filledRows = [];
        return rows;
    }

    // Движение фигуры вниз
    private moveDown(): MoveResult {
        if (this.field.canPlace(this.activeFigure!, Vec2.down())) {
            this.activeFigure!.translate(Vec2.down());
            return MoveResult.CHANGE;
        } else {
            this.field.place(this.activeFigure!);
            this.filledRows.push(...this.field.checkRows(this.activeFigure!));
            this.activeFigure = null;
            return MoveResult.PLACE;
        }
    }

    // Движение фигуры вправо/влево
    private moveLR(act: Actions.LEFT | Actions.RIGHT): MoveResult {
        let tr: Vec2 = act == Actions.LEFT ? Vec2.left() : Vec2.right();
        if (this.field.canPlace(this.activeFigure!, tr)) {
            this.activeFigure!.translate(tr);
            return MoveResult.CHANGE;
        }
        return MoveResult.CHANGELESS;
    }

    // Поворот фигуры
    private rotate(): MoveResult {
        let copy = this.activeFigure!.copy();
        this.activeFigure!.rotate(rotMat90);
        if (!this.field.canPlace(this.activeFigure!)) {
            let minPos = this.activeFigure!.min(),
                maxPos = this.activeFigure!.max();
            // Если фигура вышла за пределы поля, она перемещается обратно в
стакан,
            if (minPos.x < 0 || maxPos.x >= this.field.width) {
                this.activeFigure!.translate(new Vec2([
                    minPos.x < 0 ? -minPos.x : this.field.width - 1 - maxPos.x,
0
                ]));
```

```
            }
            // Если не удалось её выровнять, поворот отменяется
            if (!this.field.canPlace(this.activeFigure!)) {
                this.activeFigure = copy;
                return MoveResult.CHANGELESS;
            }
        }
        return MoveResult.CHANGE;
    }

    // Интерфейс управления фигурой
    move(act: Actions): MoveResult {
        if (this.activeFigure != null) {
            switch (act) {
                case Actions.DOWN:
                    return this.moveDown();
                case Actions.LEFT:
                case Actions.RIGHT:
                    return this.moveLR(act);
                case Actions.ROTATE:
                    return this.rotate();
            }
        }
        return MoveResult.EMPTY;
    }

    // Расчёт места падения фигуры
    getHint(): Figure | null {
        if (this.activeFigure) {
            let hint = this.activeFigure.copy();
            while (this.field.canPlace(hint, Vec2.down()))
                hint.translate(Vec2.down());
            return hint;
        }
        return null;
    }
}
```

### *Файл tetris/index.ts*

```
export * from "./util";
export * from "./Vec2";
export * from "./Cell";
export * from "./Figure";
export * from "./Field";
export * from "./Game";
export * from "./TetrisModel.vue"
export * from "./TetrisView.vue"
```

### *Файл tetris/TetrisModel.vue*

```
<script lang="ts">
import {Component, Emit, Prop, Vue} from 'vue-property-decorator';
import {Actions, Field, Figure, FiguresFabric, Game, MoveResult} from './index'

const _ = require('lodash');

// Интерфейс передачи состояния отображениям
export interface GameViewState {
  readonly field?: Field;
```

```
  readonly figure?: Figure | null;
  readonly hint?: Figure | null;
}

// Звуковые события
export enum SoundEvents {
  MOVE, PLACE, CLEAR, OVER
}

// Компонент логики игры
@Component
export default class TetrisModel extends Vue {
  @Prop() width!: number;
  @Prop() height!: number;
  // Базовый интервал падения
  @Prop({default: 1500}) defaultInterval!: number;
  // Множитель очков для вычитания из базового интервала
  @Prop({default: 0.3}) intervalScoreMultiplier!: number;
  // Кол-во очков за заполненный ряд
  @Prop({default: 100}) rowScore!: number;

  private game!: Game;
  private timerId: number = 0;
  private score: number = 0;
  private paused: boolean = true;

  created() {
    this.game = new Game(this.height, this.width);
    this.generateFigure();
  }

  // Вычисление интервала падения с учётом очков
  getInterval(): number {
    return this.defaultInterval - this.score * this.intervalScoreMultiplier;
  }

  // Помещение случайной фигуры в очередь
  private generateFigure() {
    this.game!.pushFigure(_.sample(FiguresFabric)())
  }

  // Спавн первой фигуры из очереди
  private spawnFigure(): boolean {
    if (this.game!.shiftFigure()) {
      this.modelChangeEmit();
      this.nextChangeEmit();
      return true;
    }
    return false;
  }

  // Авто-движение вниз
  private tick() {
    this.timerId = setTimeout(this.tick, this.getInterval());
    this.move(Actions.DOWN);
  }

  // Остановка игры
  pause() {
    if (!this.paused) {
      this.paused = true;
      clearTimeout(this.timerId);
    }
```

```
  }

  // Возобновление/старт игры
  resume() {
    if (this.paused) {
      this.paused = false;
      clearTimeout(this.timerId);
      this.timerId = setTimeout(this.tick, this.getInterval());
    }
  }

  // Интерфейс управления игрой
  move(act: Actions) {
    if (this.paused)
      return;

    let res: MoveResult = this.game!.move(act);
    switch (res) {
      case MoveResult.PLACE:
        let rows = this.game.clearFilledRows();
        this.soundEmit(SoundEvents.PLACE);
        if (rows) {
          this.score += Math.round(this.rowScore * rows * (1 + (rows - 1) /
10));
          this.scoreChangeEmit();
          this.soundEmit(SoundEvents.CLEAR);
        }
        this.modelChangeEmit();
        break
      case MoveResult.CHANGE:
        this.modelChangeEmit();
        this.soundEmit(SoundEvents.MOVE);
        break;
      case MoveResult.EMPTY:
        this.generateFigure();
        if (!this.spawnFigure()) {
          this.pause();
          this.soundEmit(SoundEvents.OVER);
          this.gameOverEmit();
        } else {
          this.soundEmit(SoundEvents.MOVE);
        }
        break;
    }
  }

  /*События игры*/

  @Emit('model-change')
  private modelChangeEmit(): GameViewState {
    return {
      field: this.game.field, figure:
      this.game.activeFigure,
      hint: this.game.getHint()
    };
  }

  @Emit('next-change')
  private nextChangeEmit(): GameViewState {
    return {figure: this.game.figuresQueue[0]};
  }

  @Emit('score-change')
```

```
  private scoreChangeEmit(): number {
    return this.score;
  }

  @Emit('game-over')
  private gameOverEmit(): boolean {
    return true;
  }

  @Emit('sound-event')
  private soundEmit(sound: SoundEvents): SoundEvents {
    return sound;
  }
}
</script>

<template>
</template>

<style scoped>
</style>
```

### *Файл tetris/TetrisView.vue*

```
<script lang="ts">
import {Component, Prop, Vue} from 'vue-property-decorator';
import {GameViewState} from "@/tetris/TetrisModel.vue";
import {Cell} from "@/tetris/Cell";

const _ = require('lodash')

// Компонент отображения игры / фигур
@Component
export default class TetrisView extends Vue {
  @Prop() width!: number;
  @Prop() height!: number;
  @Prop({default: 20}) cellSize!: number;
  @Prop({default: 1}) borderSize!: number;
  @Prop({default: '#fff'}) color!: string;
  @Prop({default: '#fff'}) bgColor!: string;
  @Prop({default: '#777'}) borderColor!: string;

  $refs!: {
    field: HTMLDivElement
    rows: HTMLDivElement[]
    cells: HTMLDivElement[]
  }

  // Обновление отображения
  update(state: GameViewState) {
    for (let cell of this.$refs.cells) {
      cell.style.backgroundColor = this.color;
      cell.style.opacity = '1';
    }

    state.hint?.cells.forEach(cell => {
      this.$refs.cells[cell.pos.y * this.width +
cell.pos.x].style.backgroundColor = cell.color;
      this.$refs.cells[cell.pos.y * this.width + cell.pos.x].style.opacity =
'0.4';
    });
```

```
    _.flatten(state.field?.mat)
        .concat(state.figure?.cells)
        .forEach(
            (cell: Cell | null) => {
              if (cell != null) {
                this.$refs.cells[cell.pos.y * this.width +
cell.pos.x].style.backgroundColor = cell.color;
                this.$refs.cells[cell.pos.y * this.width +
cell.pos.x].style.opacity = '1';
              }
            }
        );
  }
}
</script>

<template>
  <section class="field" ref="field" :style="{'background-color': bgColor}">
    <div
        class="row"
        v-for="h in height"
        ref="rows">
      <div
          class="cell"
          :style="{
            'width': `${cellSize}px`,
            'height': `${cellSize}px`,
            'background-color': color,
            'border-width' :`${borderSize}px`,
            'border-color': borderColor
          }"
          v-for="w in width"
          ref="cells">
      </div>
    </div>
  </section>
</template>

<style scoped>
.field {
  display: flex;
  flex-direction: column;
}

.row {
  display: flex;
}

.cell {
  border: solid 1px;
  transition: all linear 0.05s;
}
</style>
```

### *Файл components/Tetris.vue*

```
<script lang="ts">
import {Component, Vue} from 'vue-property-decorator';
import {SoundEvents} from "@/tetris";
import TetrisModel from "@/tetris/TetrisModel.vue";
import TetrisView from "@/tetris/TetrisView.vue";
```

```
const _ = require('lodash')

export interface Records {
  records: Array<[string, number]>;
}

@Component({
  components: {TetrisView, TetrisModel},
})
export default class Tetris extends Vue {
  width: number = 10;
  height: number = 20;
  cellSize: number = 30;
  sounds: string[] =
      ['move', 'place', 'clear', 'game_over']
          .map(name => require(`@/sounds/${name}.wav`));

  score: number = 0;
  over: boolean = false;
  playerName: string = localStorage.getItem('tetris.name') ?? '';

  $refs!: {
    model: TetrisModel,
    view: TetrisView,
    next: TetrisView,
  }

  private onKeyDown(e: KeyboardEvent) {
    if (e.keyCode >= 37 && e.keyCode <= 40)
      this.$refs.model.move(e.keyCode - 37); // сдвиг кода -37
  }

  private onResize(e?: UIEvent) {
    this.cellSize = 30 * window.innerWidth / 1920;
  }

  mounted() {
    window.addEventListener('keydown', this.onKeyDown);
    window.addEventListener('resize', this.onResize);
    this.onResize();
  }

  private beforeDestroy() {
    window.addEventListener('resize', this.onResize);
    window.removeEventListener('keydown', this.onKeyDown);
  }

  private playSound(event: SoundEvents) {
    try {
      new Audio(this.sounds[event]).play();
    } catch (e) {
      console.log('Error during playing sound');
    }
  }

  private gameOver() {
    this.over = true;
    let records = JSON.parse(localStorage.getItem('tetris.records') ??
'{"records":[]}') as Records;
    let prev = records.records.find(o => o[0] == this.playerName);
    if (prev)
      prev[1] = Math.max(this.score, prev[1]);
    else
```

```
      records.records.push([this.playerName, this.score]);
    localStorage.setItem('tetris.records', JSON.stringify(records));
  }
}
</script>

<template>
  <section class="main">
    <TetrisModel
        :width="width"
        :height="height"
        ref="model"
        @model-change="$refs.view.update($event)"
        @next-change="$refs.next.update($event)"
        @sound-event="playSound"
        @score-change="score = $event"
        @game-over="gameOver"
    />
    <transition name="game-over" mode="out-in">
      <div class="container" v-if="!over" key="game">
        <TetrisView
            ref="view"
            :width="width"
            :height="height"
            :cell-size="cellSize"
            color="#222"
            bg-color="#888"
        />

        <div class="panel">
          <div class="control">
            <button ref="resume" @click="$refs.model.resume()">Начать</button>
            <button @click="$refs.model.pause()">Пауза</button>
          </div>

          <div class="info">
            <p>Игрок : <span>{{ playerName }}</span></p>
            <p>Очки : {{ score }}</p>
          </div>

          <div class="next">
            <p>Следующая фигура:</p>
            <TetrisView
                ref="next"
                :width="4"
                :height="3"
                :border-size="1"
                border-color="#333"
                :cell-size="cellSize"
                color="#222"
            />
          </div>
        </div>
      </div>

      <div v-else class="over" key="over">
        <p>Игра окончена</p>
        <p>Очки: {{ score }}</p>
        <router-link to="/records">Перейти к рекордам</router-link>
      </div>
    </transition>

  </section>
```

```
</template>

<style scoped>
.main {
  padding: 5vh 10vw;
}

.container {
  display: flex;
  justify-content: center;
  gap: 4vw;
  transition: all 0.3s ease-in-out;
}

.panel {
  display: flex;
  flex-direction: column;
  gap: 4vh;
}

button {
  font: inherit;
  color: #ee99ee;
  text-decoration: underline #ee99ee;
  background-color: transparent;
  border: none;
  outline: none;
  transition: all 0.3s ease-in-out;
}

button:hover {
  cursor: pointer;
  transform: scale(0.97);
}

button:focus {
  text-decoration-color: #44aa44;
  color: #44aa44;
  transform: scale(0.95);
}

.next {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.info span {
  color: #44aa44;
}

.over > a {
  font-weight: bold;
  text-decoration: underline;
  color: #ee99ee;
}

.game-over-enter-active, .game-over-leave-active {
  transition: all 0.8s ease-in-out;
}

.game-over-enter {
  transform: translateX(-50%);
```

```
  opacity: 0;
}

.game-over-enter-to, .game-over-leave {
  transform: translateX(0%);
  opacity: 1;
}

.game-over-leave-to {
  transform: translateX(50%);
  opacity: 0;
}
</style>
```

### *Файл views/HomeView.vue*

```
<script lang="ts">
import {Component, Vue} from 'vue-property-decorator';

@Component
export default class HomeView extends Vue {
  name: string = '';

  $refs!: {
    input: HTMLInputElement;
  }

  mounted() {
    this.name = localStorage.getItem('tetris.name') ?? '';
  }

  start() {
    if (this.name.length > 0) {
      localStorage.setItem('tetris.name', this.name);
      this.$router.push('game');
    }
  }

  focus() {
    this.$refs.input.focus();
  }
}

</script>

<template>
  <section class="main" @click="focus">
    <div class="name-container">
      <label>Введите имя игрока >>> </label>
      <input v-model="name" autofocus ref="input">
    </div>
    <button @click="start" :class="{'active' : name.length > 0}">
      Начать игру
    </button>
  </section>
</template>

<style scoped>
.main {
  padding: 0 10vw;
  margin-top: 5vh;
}
```

```
.main > * {
  margin-top: 2vh;
}

button, input {
  font: inherit;
}

input {
  color: #44aa44;
  border: none;
  background: transparent;
  caret-color: white;
  outline: none;
}

button {
  padding: 0.5% 1%;
  border: none;
  color: #ee99ee;
  background: transparent;
  opacity: 0.5;
  transition: all ease-in-out 0.3s;
}

.active {
  opacity: 1;
  text-decoration: underline #ee99ee;
}

.active:hover {
  cursor: pointer;
}

.active:active {
  transform: scale(0.98);
}
</style>
```

### *Файл views /GameView.vue*

```
<script lang="ts">
import {Component, Vue} from 'vue-property-decorator';
import Tetris from "@/components/Tetris.vue";

@Component({
  components: {Tetris}
})
export default class GameView extends Vue {
}
</script>

<template>
  <Tetris/>
</template>

<style scoped>

</style>
```

### *Файл views /RecordsView.vue*

```
<script lang="ts">
import {Component, Vue} from 'vue-property-decorator';
import {Records} from "@/components/Tetris.vue";

@Component
export default class RecordsView extends Vue {
  records!: Records;

  created() {
    this.records = JSON.parse(localStorage.getItem('tetris.records') ??
'{"records":[]}') as Records;
    this.records.records.sort((a, b) => b[1] - a[1]);
  }
}

</script>

<template>
  <section class="main">
    Таблица рекордов
    <hr>
    <table>
      <tr v-for="rec of records.records">
        <td class="name">{{ rec[0] }}</td>
        <td class="score">{{ rec[1] }}</td>
      </tr>
    </table>
  </section>
</template>

<style scoped>
.main {
  margin-top: 5vh;
  padding: 0 10vw;
}

table {
  text-align: center;
  margin: auto;
  border-spacing: 1vh;
}

.name {
  color: #44aa44;
}

.score {
  font-weight: bold;
  color: #ee99ee;
}
</style>
```

### *Файл router/index.ts*

```
import Vue from 'vue';
import VueRouter, {RouteConfig} from 'vue-router';

Vue.use(VueRouter);

const routes: Array<RouteConfig> = [
```

```
    {
        path: '/',
        name: 'home',
        component: () => import('../views/HomeView.vue')
    },
    {
        path: '/game',
        name: 'game',
        component: () => import('../views/GameView.vue')
    },
    {
        path: '/records',
        name: 'records',
        component: () => import('../views/RecordsView.vue')
    }
];

const router = new VueRouter({
    routes
});

export default router;
```

### *Файл App.vue*

```
<template>
  <div id="app">
    <nav>
      <router-link to="/">Игра</router-link>
      |
      <router-link to="/records">Рекорды</router-link>
    </nav>
    <router-view/>
  </div>
</template>

<style>
body {
  background-color: #222222
}

#app {
  font-family: "Lucida Console", sans-serif;
  text-align: center;
  font-size: 2vw;
  color: #dadada;
}

nav {
  padding: 3vh 3vw;
}

nav a {
  font-weight: bold;
  text-decoration: none;
  color: inherit;
}

nav a.router-link-exact-active {
  color: #44aa44;
}
</style>
```

### *Файл main.ts*

```
import Vue from 'vue';
import App from './App.vue';
import router from './router';

Vue.config.productionTip = false;

new Vue({
    router,
    render: h => h(App)
}).$mount('#app');
```

### *Файл conf.nginx*

```
server {
   listen 80;
   server_name localhost;
   return 301 https://localhost;
}

server {
   listen 443 ssl http2;
   server_name localhost;

   #SSL settings
   ssl_certificate /home/vlad/ssl/cert.crt;
   ssl_certificate_key /home/vlad/ssl/key.key;

   charset utf-8;
   root /home/vlad/WebstormProjects/WebLabs/dist;
   index index.html;
}
```