Руководство пользователя:

1 Страница:

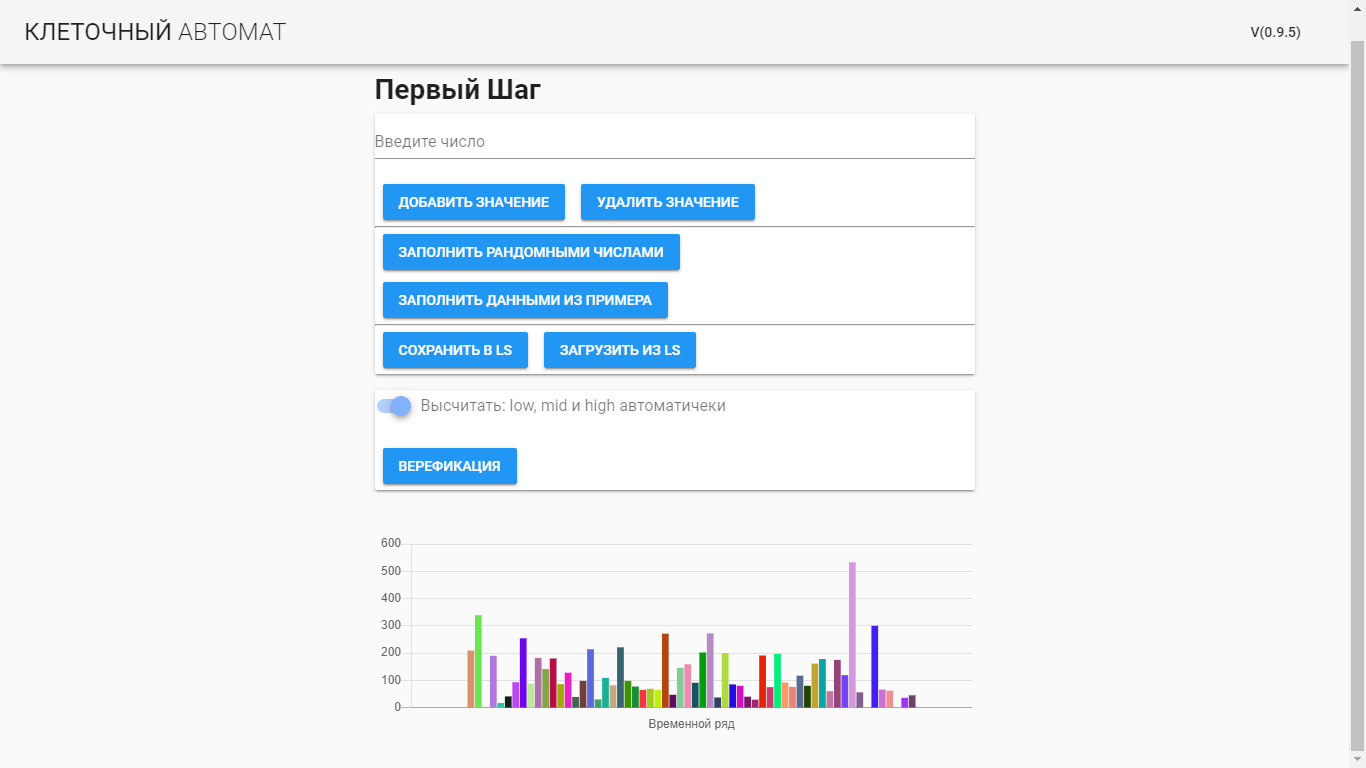


Рис 1 – первая страница.

Кнопки:

Добавить – добавляет один 1 столбец.

Удалить – удаляет один 1 столбец.

Заполнить рандомными числами – заполняет график 60 числами (от 12 до 30).

Заполнить данными из примеа– заполняет график 60 числами.

Сохранить в ls(localStorage) – сохраняет исходные данные в браузере.

Загрузить из ls – загружает заранее сохраненные данные из памяти.

Верефикация – отправляет данные на верефикацию.

Чекбокс - Позволяет вводить данные вершины минимума и вершины среднего значения.

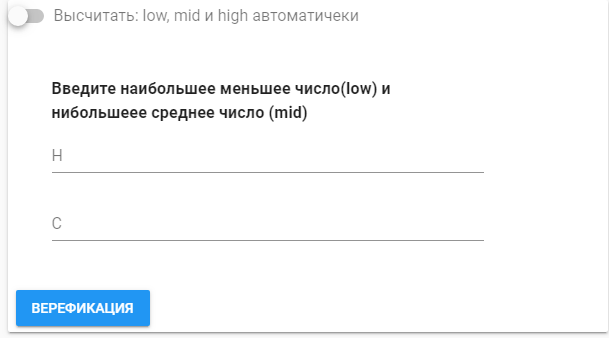


Рис 2 – ввод вручную.

вторая страница:

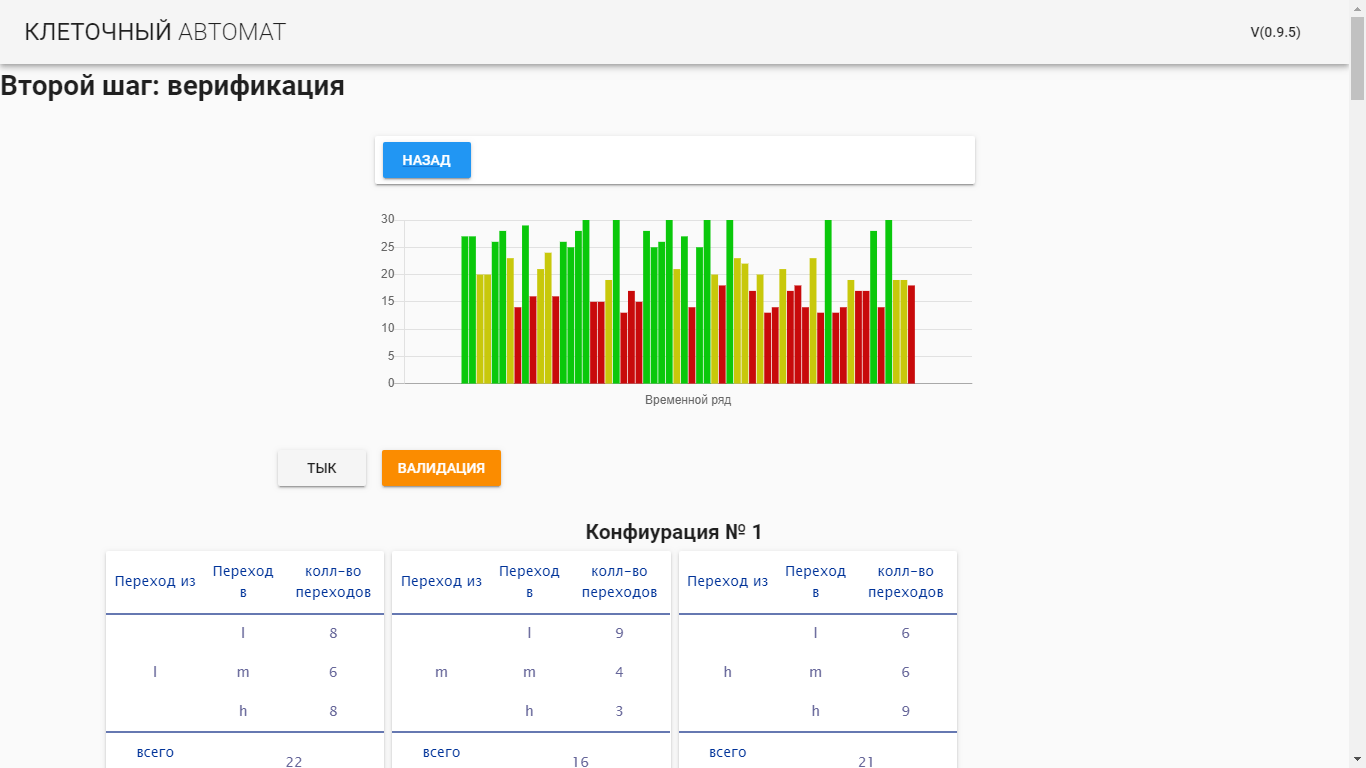


Рис 3 – этап верификации.

Кнопки:

Назад – возвращает назад (на 1 страницу).

Тык – Костыль который заново проводит верефикацию.

Валидация – переход на следущий этап – этап валидации.

Третья страница:

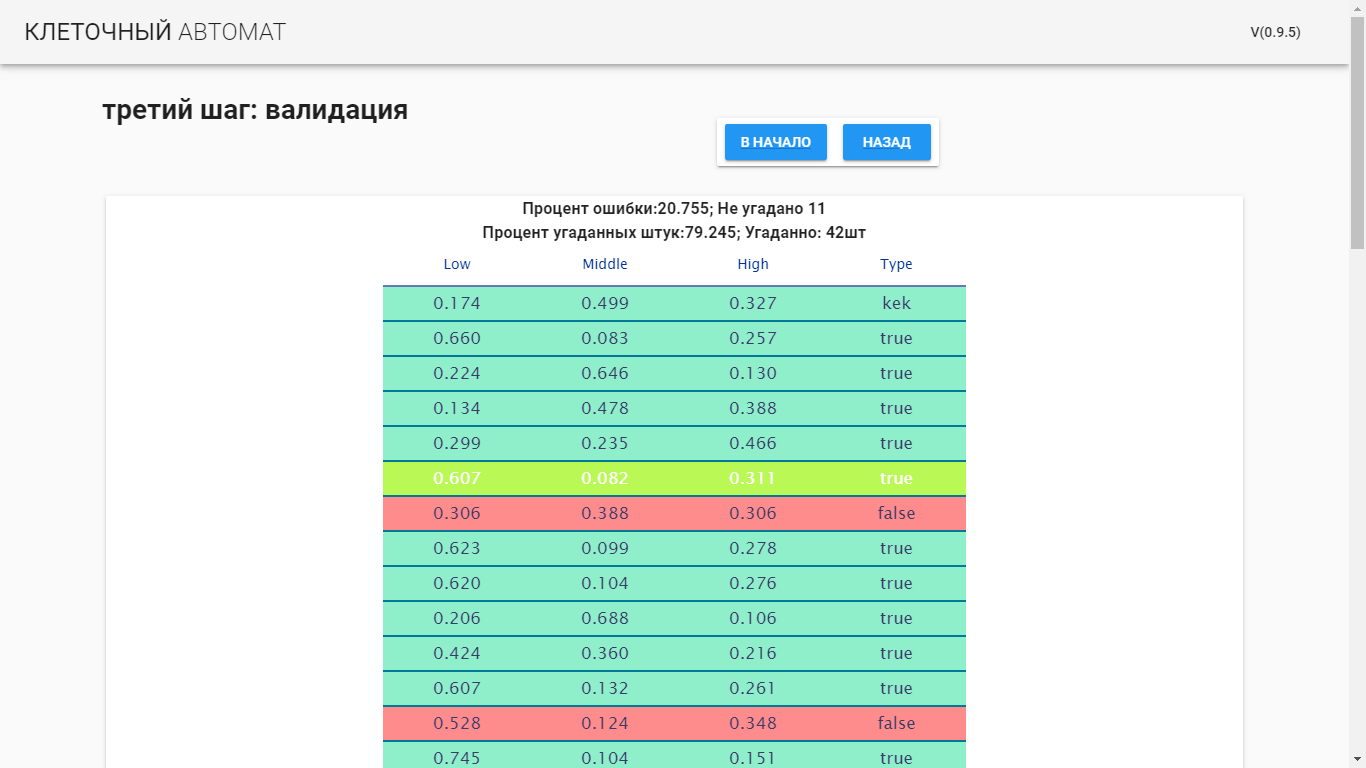


Рис 4 – этап валидации.

Кнопки:

В начало – возвращает на 1 страницу.

Назад – возвращает на этап верификации.

Листинг:

Основные функции верификации :

methods: {

async newArr() {

let some = this.some;

*// console.log("some", some);*

let adaptiveArray = ["l", "m", "h"]; *// массив с параметрами*

let catchMap = new Map();

for (const item in adaptiveArray) {

let returnedObj;

let l = 0;

let m = 0;

let h = 0;

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

if (some[i] == adaptiveArray[item]) {

switch (some[i + 1]) {

case "l":

l++;

break;

case "m":

m++;

break;

case "h":

h++;

break;

}

}

}

*// console.log(`l: ${l}, m: ${m} h ${h} `);*

returnedObj = {

l: l,

m: m,

h: h

};

catchMap.set(adaptiveArray[item], returnedObj);

}

this.validArray = await [catchMap];

this.MoveToNewConfiguration();

},

fart(){

this.MoveToNewConfiguration();

this.snackbar = true;

},

saveChanges(){

this.$store.commit('setVerificArray', this.verArray)

this.$store.commit('setDeep', this.validArray.length)

this.$router.push('/third');

},

MoveToNewConfiguration() {

const lastMap = this.validArray[length - 1].entries();

*// console.log('lm ',this.validArray[length - 1].size )*

if (this.validArray[length - 1].size != 0) {

*// console.log(*

*// `Начало работы с конфигурацией № ${this.validArray.length + 1}`*

*// );*

let catchMap = new Map();

let rate = ["l", "m", "h"];

for (const arr of lastMap) {

*//проверка на то продолжать ли увеличивать конфигурацию с этой моделью*

let itertator = 0;

let overlook = [];

arr[1].l != 0 ? itertator++ : overlook.push("l");

arr[1].m != 0 ? itertator++ : overlook.push("m");

arr[1].h != 0 ? itertator++ : overlook.push("h");

*// console.log(arr[1], ' ', overlook);*

if (itertator != 1 && itertator != 0) {

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

let element = rate[i];

if (overlook.indexOf(element) == -1) {

let newCH = `${arr[0]}${element}`;

let returnedObj;

let l = 0;

let m = 0;

let h = 0;

let some = this.some.join("");

let lengthOfNewCH = newCH.length;

let regexp = new RegExp(newCH, "g");

let errorCheck = false;

*// console.log('newch',newCH)*

let lengthOfDeep = some.match(regexp);

if (lengthOfDeep != null) {

lengthOfDeep = lengthOfDeep.length;

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

*// ll => llm,lll,llh*

let index = some.search(regexp);

some = some.split("");

some.splice(index, lengthOfNewCH);

some = some.join("");

switch (some[index]) {

case "l":

l++;

break;

case "m":

m++;

break;

case "h":

h++;

break;

}

}

*// console.log(`CH = ${newCH} => l: ${l}, m: ${m} h ${h} `);*

returnedObj = {

l: l,

m: m,

h: h

};

if(returnedObj.l + returnedObj.m + returnedObj.h != 0){

catchMap.set(newCH, returnedObj);

}

}

}

}

}

}

this.validArray = [catchMap, ...this.validArray];

this.MoveToNewConfiguration();

} else {

this.validArray.shift();

*// console.log(*

*// `Максимальный уовень конфигурации: ${this.validArray.length}`*

*// );*

*// console.log(Array.from(this.validArray[0]))*

let finalArray = new Array();

this.verArray = this.validArray;

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

finalArray.push(Array.from(this.validArray[i]));

}

this.validArray = finalArray.reverse();

}

},

allMoves(*o*) {

return Object.keys(o).reduce(function(*previous*, *key*) {

return previous + o[key];

}, 0);

}

}

Основные функции валидации:

{

changeBool(*item*){

console.log(item.bool)

item.bool = !item.bool

this.getPercent()

},

getPercent() {

let correctnessArray = this.boolArray.reduce(function(*acc*, *el*) {

acc[el] = (acc[el] || 0) + 1;

return acc;

}, {});

correctnessArray = {

percent: (correctnessArray.true\*100/(correctnessArray.true+correctnessArray.false)).toFixed(3),

error: (100-correctnessArray.true\*100/(correctnessArray.true+correctnessArray.false)).toFixed(3)

,

...correctnessArray

}

this.correctness = correctnessArray

},

findBlah() {

let mainObj = this.mainArray;

let verArray = this.verArray;

*// verArray = verArray.reverse();*

let deep = this.deep;

let str = mainObj[length - 1];

let mainArray = new Array();

let mainArraySec = new Array();

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

mainArray.push(mainObj[i].type);

mainArraySec.push(mainObj[i].type);

}

this.counter = mainArray.length;

this.mainArray = mainArraySec;

let MALength = mainArray.length;

*// console.log(`MALength ${MALength}, deep ${deep}`)*

let counter = 0;

*// for (let index = 0; index < 7; index++) {*

for (let index = 0; index < MALength; index++) {

let lastTypes = [];

counter++;

lastTypes = mainArray.slice(-deep);

mainArray.pop();

let toLow = [],

toMiddle = [],

toHigh = [],

l = null,

m = null,

h = null;

for (let i = 1; i <= lastTypes.length; i++) {

*//выводим последние элементs*

let types = lastTypes.slice(-i).join("");

let map = verArray[verArray.length - i];

*// console.log(map, types)*

if (map.has(types)) {

let cases = map.get(types); *//это l m h со значениеями от даннго types*

*// console.log('kek')*

let sum = cases.l + cases.m + cases.h;

toLow.push(cases.l / sum);

toMiddle.push(cases.m / sum);

toHigh.push(cases.h / sum);

*// console.log('cases: ', cases )*

} else {

console.error(

"error, dats table isn't exist, it's normal -- calm down!"

);

*// return null*

}

}

let forecast = this.makeForecast(toLow, toMiddle, toHigh);

*// console.log('arrays: ', forecast)*

this.counter--;

this.validArray.push(forecast);

*// this.mainArray.pop()*

*//*

}

return this.validArray;

},

makeForecast(*l*, *m*, *h*) {

let low = 0,

middle = 0,

high = 0;

for (let index = 0; index < l.length; index++) {

let sum = l[index] + m[index] + h[index];

low = low + l[index];

middle += m[index];

high += h[index];

}

let summ = low + middle + high;

*// if (this.counter != ) {*

*// }*

*// console.log(this.mainArray.length+1, this.counter)*

if (this.counter == this.mainArray.length || this.counter <= this.deep) {

*// this.mainArray.pop()*

this.getPercent()

return {

*// counter: this.counter,*

l: (low / summ).toFixed(3),

m: (middle / summ).toFixed(3),

h: (high / summ).toFixed(3),

bool: "kek"

};

} else {

let l = low / summ,

m = middle / summ,

h = high / summ;

let bool = new Boolean();

let maximum = Math.max(l, m, h);

switch (maximum) {

case l:

maximum = "l";

break;

case m:

maximum = "m";

break;

case h:

maximum = "h";

break;

}

console.log(

"maximum: ",

maximum,

", mainArray{index}: ",

this.mainArray[this.counter],

this.counter

);

maximum == this.mainArray[this.counter]

? (bool = true)

: (bool = false);

if (bool == true) {

this.boolArray.push(true);

} else {

this.boolArray.push(false);

}

return {

l: (low / summ).toFixed(3),

m: (middle / summ).toFixed(3),

h: (high / summ).toFixed(3),

bool: bool

};

}

}

}

Ссылка на весь проект: <https://github.com/Hakoba/cellular-automaton>