# ДОДАТОК А

Код розробленої програмної системи трасування маршрутів

<!DOCTYPE HTML>

<html>

<head>

<title>Danylo Shevchuk Bachelor</title>

<link rel="stylesheet" href="foundation-icons.css"/>

<meta charset="utf-8">

<style>

table td {

padding: 5px;

border: 1px solid black;

width: 30px;

height: 30px;

text-align: center;

font-size: 100%;

}

input.bonus {

width: 45px;

}

input.penalty {

width: 45px;

}

p.h {

font-weight: bold;

}

p {

margin: 5px;

}

div {

padding: 5px;

border: 1px solid black;

margin: 5px;

}

td#empty {

background-color: #ffffff;

}

td#wall {

background-color: #aa872a;

}

td#accelerator {

background-color: #5da92e;

}

td#slower {

background-color: #ffdc00;

}

td#start {

background-color: #0000ff;

}

td#finish {

background-color: #ee2426;

}

td#path {

background-color: rgba(31, 31, 31, 0.79);

color: #ffffff;

}

i {

font-size: 18px;

}

select option#hidden {

display: none;

}

</style>

<!--[if lte IE 8]>

<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script><![endif]-->

</head>

<body>

<div id='tableNavigator'>

<div id='tableContainer' style="border: 0; padding: 0; margin: 0">

<p class='h'> Map </p>

</div>

<p class="h"> Change Map Scale </p>

<input type='button' class='scale' id='plus' value='+'/> <input type='button' class='scale' id='minus' value='-'/>

<br>

<p class="h"> Create, Export or Import Map </p>

<input type='button' class='file' id='newMap' value='Create New Map'/> <input type='button' class='file'

id='saveMap'

value='Export Map to file...'

title="Wave indexes will not save, so you can re-use this map after re-uploading it"

disabled/>

<p> Or just Import your pre-made map! </p>

<input type='file' class='file' id='loadMapImport' value='Import Map from .json file...' accept=".json"/>

<br>

<input type='button' class='file' id='loadMap' value='Load Map...' style="margin-top: 4px"/>

</div>

<div>

<p id='modSelectorContainer'>

<p class='h'> Edit mode selector </p>

<p><input type='radio' name='mod' value='empty' checked> Empty <i class="trash" style="color: #000000"></i></p>

<p><input type='radio' name='mod' value='wall'> Wall <i class="align-justify" style="color: #aa872a"></i></p>

<p><input type='radio' name='mod' value='accelerator'> Accelerator <i class="fast-forward" style="color: #5da92e"></i> cell bonus - <input type="number" step=".01" min=".1" max=".9" id="accelerator" value=".5" class="bonus"></p>

<p><input type='radio' name='mod' value='slower'> Slower <i class="mountains" style="color: #ffdc00"></i> cell penalty + <input type="number" step=".01" min=".1" max=".9" id="slower" value=".5" class="penalty"></p>

<p><input type='radio' name='mod' value='start'> Start <i class="home" style="color: #0000ff"></i></p>

<p><input type='radio' name='mod' value='finish'> Finish <i class="flag" style="color: #ee2426"></i></p>

</div>

<div id='buttonsContainer'>

<p class='h'> Route tracing </p>

<input type="button" id="start" value="Start" disabled/>

<p><input type='checkbox' name='step' value='finish'> Step-by-step mode</p>

<p><input type='checkbox' name='direction' value='finish'> 8 directional mode</p>

<input type="button" id="next" value="Next Step" disabled/>

</div>

<script>

const mods = ['empty', 'wall', 'accelerator', 'slower', 'start', 'finish', 'path'];

const modsContent = [false, 'align-justify', 'fast-forward', 'mountains', 'home', 'flag'];

let config = {

mapSize: {

x: 7,

// x: prompt('Enter the heights of the map you want to create:', '7') || 1,

// y: prompt('Enter the width of the map you want to create:', '7') || 1,

y: 7,

z: 4

}

};

let twoDim = true;

let startPosition = null;

let finishPosition = null;

let map = [];

let modButtons = document.getElementsByName('mod');

let currentMod = 'empty';

let acceleratorBonus = 0.5;

let slowerPenalty = 0.5;

initModSelector();

let stepMode = false;

let diagonalMode = false;

const startButton = document.getElementById('start');

initStarter();

let currentScale = 10;

let currentCellStyle = {

width: '30px',

height: '30px',

fontSize: '100%',

};

initScaleMenu();

initSaveMenu();

let table;

function init(newMapNeeded) {

startButton.disabled = false;

if (newMapNeeded) {

map = [];

initMap();

}

updateTable(map);

}

function initModSelector() {

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

modButtons[i].onchange = selectChange;

}

function selectChange() {

currentMod = this.value;

}

document.getElementById('accelerator').onchange = acceleratorOnChange;

function acceleratorOnChange() {

acceleratorBonus = this.value;

}

document.getElementById('slower').onchange = slowerOnChange;

function slowerOnChange() {

slowerPenalty = this.value;

}

}

function initStarter() {

startButton.onclick = () => {

if (checkStartEndReady()) {

stepSwitches[0].disabled = true;

directionSwitches[0].disabled = true;

document.getElementById('accelerator').disabled = true;

document.getElementById('slower').disabled = true;

if (stepMode) {

nextButton.disabled = !stepMode;

nextButton.click();

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

modButtons[i].disabled = true;

}

} else {

waveStart()

}

}

};

const nextButton = document.getElementById('next');

// nextButton.onclick = nextStep();

nextButton.onclick = () => {

if (checkStartEndReady()) {

startButton.disabled = true;

stepSwitches[0].disabled = true;

directionSwitches[0].disabled = true;

nextStep();

}

};

const stepSwitches = document.getElementsByName('step');

stepSwitches[0].addEventListener('change', () => {

stepMode = !stepMode;

console.log('stepMode ===', stepMode)

});

const directionSwitches = document.getElementsByName('direction');

directionSwitches[0].addEventListener('change', () => {

diagonalMode = !diagonalMode;

console.log('diagonalMode ===', diagonalMode)

});

}

function initScaleMenu() {

const plus = document.getElementById('plus');

plus.onclick = () => {

currentScale += 1;

if (currentScale >= 20) {

plus.disabled = true;

}

if (currentScale > 1) {

minus.disabled = false;

}

applyScale();

};

const minus = document.getElementById('minus');

minus.onclick = () => {

currentScale -= 1;

console.log('new scale', currentScale);

if (currentScale < 20) {

plus.disabled = false;

}

if (currentScale <= 1) {

minus.disabled = true;

}

applyScale();

};

}

function initSaveMenu() {

const newMap = document.getElementById('newMap');

newMap.onclick = () => {

const x = askToEnterSize('Enter the heights of the map you want to create:');

const y = askToEnterSize('Enter the width of the map you want to create:');

config = {

mapSize: {

x: x,

y: y,

z: 4

}

};

newMap.disabled = true;

loadMap.disabled = true;

saveMap.disabled = false;

loadMapImport.disabled = true;

init(true);

};

const loadMapImport = document.getElementById('loadMapImport');

const loadMap = document.getElementById('loadMap');

loadMap.onclick = function () {

let files = loadMapImport.files;

console.log(files);

if (files.length <= 0) {

alert('Chose files first!');

return false;

}

let fr = new FileReader();

fr.onload = function (e) {

let result = JSON.parse(e.target.result);

console.log(result);

if (result.config && result.savedMap) {

map = result.savedMap;

config = result.config;

finishPosition = result.finishPosition;

startPosition = result.startPosition;

newMap.disabled = true;

loadMap.disabled = true;

saveMap.disabled = false;

loadMapImport.disabled = true;

init(false);

console.log(map);

} else {

alert('There is something wrong with your file!')

}

};

fr.readAsText(files.item(0));

};

const saveMap = document.getElementById('saveMap');

saveMap.onclick = () => {

let mapToSave = Array.from(map);

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

const column = mapToSave[i];

for (let j = 0; j < column.length; j++) {

const cell = column[j];

if (cell.data?.arrow) {

column[j] = {

mode: cell.mode,

position: cell.position,

content: modsContent[cell.mode],

// waveIndex: null,

}

} else {

column[j] = {

mode: cell.mode,

position: cell.position,

content: cell.content,

// waveIndex: null,

}

}

}

}

let dataStr = "data:text/json;charset=utf-8," + encodeURIComponent(JSON.stringify({

savedMap: mapToSave,

config: {

mapSize: {

x: config.mapSize.x,

y: config.mapSize.y

}

},

finishPosition: finishPosition,

startPosition: startPosition

}));

let downloadAnchorNode = document.createElement('a');

downloadAnchorNode.setAttribute("href", dataStr);

downloadAnchorNode.setAttribute("download", "saved\_map.json");

document.body.appendChild(downloadAnchorNode); // required for firefox

downloadAnchorNode.click();

downloadAnchorNode.remove();

};

}

function askToEnterSize(msg) {

let size = Number(prompt(msg, '7'));

while (!size || size % 1 || size <= 0 || size > 126) {

size = Number(prompt('Please try other number', '7'));

}

return size;

}

function applyScale() {

currentCellStyle.width = 30 \* currentScale / 10 + 'px';

currentCellStyle.height = 30 \* currentScale / 10 + 'px';

currentCellStyle.fontSize = currentScale \* 10 + '%';

currentCellStyle.iconSize = 18 \* currentScale / 10 + 'px';

updateTable(map);

}

function initMap() {

for (let i = 0; i < config.mapSize.x; i++) {

const row = [];

for (let j = 0; j < config.mapSize.y; j++) {

row.push({

mode: 0,

position: {

x: i,

y: j

}

});

}

map.push(row);

}

}

function updateTable(twoDimArray) {

table = getNewTable();

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

const column = twoDimArray[i];

const tr = document.createElement('tr');

for (let j = 0; j < column.length; j++) {

const cell = column[j];

const td = document.createElement('td');

td.style.width = currentCellStyle.width;

td.style.height = currentCellStyle.height;

td.style.fontSize = currentCellStyle.fontSize;

td.id = cell.data?.arrow ? mods[6] : mods[cell.mode];

if (cell.content) {

if (cell.waveIndex) {

if (cell.data?.arrow) {

if (cell.mode === 2) {

updateCell(td, '', '#5da92e');

addIcon(td, cell);

} else if (cell.mode === 3) {

updateCell(td, '', '#ffdc00');

addIcon(td, cell);

} else {

updateCell(td, '');

addIcon(td, cell);

}

} else {

if (cell.mode === 2) {

updateCell(td, cell.waveIndex, cell.data?.arrow ? '#5da92e' : null);

} else if (cell.mode === 3) {

updateCell(td, cell.waveIndex, cell.data?.arrow ? '#ffdc00' : null);

} else {

updateCell(td, '');

addIcon(td, cell);

}

}

} else {

updateCell(td, '');

addIcon(td, cell);

}

} else {

updateCell(td, cell.waveIndex);

}

td.addEventListener('click', function tdClick() {

setCell(this, cell);

});

tr.appendChild(td)

}

table.appendChild(tr)

}

}

function updateCell(td, waveIndex, color) {

if (waveIndex > 0) td.innerHTML = waveIndex % 1 ? (+waveIndex).toFixed(1) : waveIndex;

if (color) td.style.color = color;

}

function setCell(td, cell) {

if (waveStarted) return;

if (currentMod === 'start' && startPosition) {

alert('There should be only one start, please select other block')

} else if (currentMod === 'finish' && finishPosition) {

alert('There should be only one finish, please select other block')

} else {

if (currentMod === 'start') {

startPosition = {x: cell.position.x, y: cell.position.y};

} else if (currentMod === 'finish') {

finishPosition = {x: cell.position.x, y: cell.position.y};

}

if (td.id === 'start') {

startPosition = null;

} else if (td.id === 'finish') {

finishPosition = null;

}

td.id = currentMod;

updateCell(td, cell.waveIndex);

const index = mods.indexOf(currentMod);

cell.mode = index;

cell.content = modsContent[index] || null;

const icon = td.querySelectorAll('i');

if (icon[0]) {

icon[0].remove();

}

addIcon(td, cell);

}

}

function addIcon(td, cell) {

const icon = document.createElement('i',);

icon.className = cell.content;

icon.style.fontSize = currentCellStyle.iconSize;

td.appendChild(icon);

}

function clearTable() {

table = document.getElementById('table');

if (table) {

table.remove();

// parent.delete(table);

}

}

function getNewTable() {

const parent = document.getElementById('tableContainer');

clearTable(parent);

let table = document.createElement('table');

table.id = 'table';

parent.appendChild(table);

return table;

}

// wave algo section! BEWARE!!!

let elementsQueue = [];

let nextElements = [];

let biggestWaveIndex = 0;

let finishFound = false;

let wayHomeFound = false;

let currentCell = null;

// step mode variables

let waveStarted = false;

let pathFinderStarted = false;

function waveNeighbor(x, y, nextWaveIndex) {

if (map[x] && map[x][y]) {

const mode = map[x][y].mode;

if (mode === 0 && (!map[x][y].waveIndex || map[x][y].waveIndex > +nextWaveIndex.toFixed(1))) {

map [x][y].waveIndex = +nextWaveIndex.toFixed(1);

nextElements.push({x: x, y: y})

} else if (mode === 2 && (!map[x][y].waveIndex || map[x][y].waveIndex > +nextWaveIndex.toFixed(1) - +acceleratorBonus)) {

map [x][y].waveIndex = +nextWaveIndex.toFixed(1) - +acceleratorBonus;

nextElements.push({x: x, y: y})

} else if (mode === 3 && (!map[x][y].waveIndex || map[x][y].waveIndex > +nextWaveIndex.toFixed(1) + +slowerPenalty)) {

map [x][y].waveIndex = +nextWaveIndex.toFixed(1) + +slowerPenalty;

nextElements.push({x: x, y: y})

} else if (mode === 5) {

finishFound = true;

biggestWaveIndex = map [x][y].waveIndex = +nextWaveIndex.toFixed(1);

}

}

}

function waveStep(position) {

const x = position.x;

const y = position.y;

map[x][y].waveIndex = map[x][y].waveIndex || 0;

const currentWaveIndex = map[x][y].waveIndex;

[

[x + 1, y],

[x - 1, y],

[x, y + 1],

[x, y - 1],

].forEach(el => {

waveNeighbor(el[0], el[1], currentWaveIndex + 1);

if (finishFound) return false;

});

if (diagonalMode) {

[

[x + 1, y - 1],

[x - 1, y + 1],

[x + 1, y + 1],

[x - 1, y - 1],

].forEach(el => {

waveNeighbor(el[0], el[1], currentWaveIndex + 1.4);

if (finishFound) return false;

});

}

}

function checkStartEndReady() {

if (!startPosition) {

alert('Please mark starting position before running');

return false;

} else {

if (!finishPosition) {

alert('Please mark finish position before running');

return false;

} else return true;

}

}

function waveStart() {

if (checkStartEndReady()) {

waveStep(startPosition);

// currentWaveIndex++;

waveStarted = true;

while (nextElements[0]) {

elementsQueue = nextElements;

nextElements = [];

while (elementsQueue[0]) {

waveStep(elementsQueue[0]);

elementsQueue.splice(0, 1);

if (finishFound) break;

}

// currentWaveIndex++;

updateTable(map);

console.log('step ended');

if (finishFound) {

// currentBackIndex = currentWaveIndex;

// currentWaveIndex = 0;

break

}

}

console.log('wave ended');

if (finishFound) {

findWayHome();

} else {

alert('Sadly to say, but I do not know da wae.');

alert('But you can save the map and re-upload it later!');

}

}

}

function comeBackStep(position) {

const arrowIconConfig = [

'arrow-up',

'arrow-down',

'arrow-left',

'arrow-right',

// diagonal mode

'arrow-up-right',

'arrow-down-left',

'arrow-up-left',

'arrow-down-right',

];

let neighborIndex;

let lowestNeighbor;

const x = position.x;

const y = position.y;

const currentBackIndex = map[x][y].waveIndex > -1 ? map[x][y].waveIndex : biggestWaveIndex;

if (map[x + 1] && map[x + 1][y] && map[x + 1][y].waveIndex > -1 && map [x + 1][y].waveIndex < currentBackIndex) {

neighborIndex = 0;

lowestNeighbor = map[x + 1][y];

}

if (map[x - 1] && map[x - 1][y] && map[x - 1][y].waveIndex > -1 && map [x - 1][y].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex : currentBackIndex)) {

neighborIndex = 1;

lowestNeighbor = map[x - 1][y];

}

if (map[x] && map[x][y + 1] && map[x][y + 1].waveIndex > -1 && map [x][y + 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex : currentBackIndex)) {

neighborIndex = 2;

lowestNeighbor = map[x][y + 1];

}

if (map[x] && map[x][y - 1] && map[x][y - 1].waveIndex > -1 && map [x][y - 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex : currentBackIndex)) {

neighborIndex = 3;

lowestNeighbor = map[x][y - 1];

}

if (diagonalMode) {

if (map[x + 1] && map[x + 1][y - 1] && map[x + 1][y - 1].waveIndex > -1 && map [x + 1][y - 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex : currentBackIndex)) {

neighborIndex = 4;

lowestNeighbor = map[x + 1][y - 1];

}

if (map[x - 1] && map[x - 1][y + 1] && map[x - 1][y + 1].waveIndex > -1 && map [x - 1][y + 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex : currentBackIndex)) {

neighborIndex = 5;

lowestNeighbor = map[x - 1][y + 1];

}

if (map[x + 1] && map[x + 1][y + 1] && map[x + 1][y + 1].waveIndex > -1 && map [x + 1][y + 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex : currentBackIndex)) {

neighborIndex = 6;

lowestNeighbor = map[x + 1][y + 1];

}

if (map[x - 1] && map[x - 1][y - 1] && map[x - 1][y - 1].waveIndex > -1 && map [x - 1][y - 1].waveIndex < (lowestNeighbor ? lowestNeighbor.waveIndex : currentBackIndex)) {

neighborIndex = 7;

lowestNeighbor = map[x - 1][y - 1];

}

}

if (lowestNeighbor && lowestNeighbor.mode !== 4) {

lowestNeighbor.content = arrowIconConfig[neighborIndex];

lowestNeighbor.data = {arrow: true};

}

return lowestNeighbor;

}

function findWayHome() {

currentCell = map[finishPosition.x][finishPosition.y];

while (currentCell && currentCell.mode !== 4) {

currentCell = comeBackStep(currentCell.position);

console.log('step back found');

updateTable(map);

// currentBackIndex--;

}

wayHomeFound = true;

startButton.disabled = true;

console.log('way home found');

console.log(map);

}

function nextStep() {

if (wayHomeFound) {

alert('refresh page to start over')

} else {

if (finishFound) {

if (pathFinderStarted) {

findWayHomeStep();

} else {

currentCell = map[finishPosition.x][finishPosition.y];

pathFinderStarted = true;

findWayHomeStep();

}

} else {

if (waveStarted) {

continueWave();

} else {

startWaveStep()

}

}

}

}

function startWaveStep() {

waveStep(startPosition);

waveStarted = true;

updateTable(map);

}

function continueWave() {

if (nextElements[0]) {

elementsQueue = nextElements;

nextElements = [];

while (elementsQueue[0]) {

waveStep(elementsQueue[0]);

elementsQueue.splice(0, 1);

if (finishFound) break;

}

updateTable(map);

console.log('step ended');

if (finishFound) {

nextElements = [];

}

} else {

if (!finishFound) {

alert('Sadly to say, but I do not know da wae.');

alert('But you can save the map and re-upload it later!');

wayHomeFound = true;

}

}

}

function findWayHomeStep() {

if (currentCell && currentCell.mode !== 4) {

currentCell = comeBackStep(currentCell.position);

console.log('step back found');

updateTable(map);

// currentBackIndex--;

} else {

wayHomeFound = true;

startButton.disabled = true;

console.log('way home found');

// console.log(map);

}

}

</script>

</body>

</html>