****

**实 验 报 告**

|  |  |
| --- | --- |
| **实验课程：** | **人工智能导论** |
| **实验项目：** | **八数码问题** |
| **指导教师：** |  |
| **学生姓名:** |  |
| **学生学号：** |  |
| **班 级：** |  |
| **分 数：** |  |

20 年 月 日

成都信息工程大学 计算机学院

## 问题描述

实现八数码问题的求解，要求实现一种盲目搜索算法和一种启发式搜索算法，并比较两种算法的求解效率，包括搜索空间大小和解的步数。使用可视化的界面演示求解过程，并建议对于任意初始状态进行是否有解的判断。

## 解决方案阐述

Vscode编写vue和ts，Vscode上F5运行，在浏览器上交互，bfs实现盲目搜索，初始状态到某个状态再到目标状态的步数作为启发函数来实现启发式搜索

## 设计算法描述

盲目搜索算法：

Bfs搜索所有初始状态能到达的状态，直到搜索出目标状态

启发式搜索算法：

定义启发函数f(x)表示经过x这个状态，到达目标状态的估计最小步数

f(x)=g(x)+h(x)，g(x)表示从初始状态到x这个状态的最小步数，h(x)表示从x这个状态到目标状态的估计最小步数

h(x)是x这个状态里，所有数字到目标状态的曼哈顿距离之和

把初始状态丢到优先队列里，每次从优先队列里取出f最小的状态，计算该状态可以到达而且之前没有到达的状态的f，并将这些后继状态入队，直至搜索出目标状态

因为得出的启发函数是松弛问题的确切代价，那么它一定遵守三角不等式，因而是一致的，所以有可采纳性

## 算法实现（即完整源程序，带注解）

<script lang="ts" setup></script>

<style lang="scss" scoped>

.author-card {

display: flex;

align-items: center;

justify-content: center;

width: 240px;

height: 22px;

padding: 12px;

margin-top: 12px;

font-size: 14px;

background-color: #fff;

border-radius: 6px;

box-shadow: 2px 2px 3px rgb(0 0 0 / 20%);

img {

height: 100%;

}

p {

margin-left: 6px;

}

}

</style>

<script lang="ts" setup>

import { Stage, type GlobalState } from '@/types';

import { computed, inject } from 'vue';

import OneChecker from './OneChecker.vue';

const state = inject('state') as GlobalState;

const showSolved = computed(() => {

if (state.stage === Stage.ManualControl && state.current === state.target) {

return true;

} else {

return false;

}

});

const showPlaySolution = computed(() => {

if (state.stage === Stage.UninformedSearchEnd && state.solution.length > 0) {

return true;

} else if (state.stage === Stage.AStarSearchEnd && state.solution.length > 0) {

return true;

} else {

return false;

}

});

const handlePlaySolution = async () => {

state.stage = Stage.PlayingSolution;

for (const item of state.solution) {

state.current = item;

// Sleep 750ms

const sleep750ms = new Promise<void>((resolve) => {

setTimeout(() => {

resolve();

}, 750);

});

await sleep750ms;

}

state.stage = Stage.Solvable;

};

const showNotSolvable = computed(() => {

if (state.stage === Stage.NotSolvable) {

return true;

} else if (state.stage === Stage.UninformedSearchEnd && state.steps === -1) {

return true;

} else if (state.stage === Stage.AStarSearchEnd && state.steps === -1) {

return true;

} else {

return false;

}

});

</script>

<template>

<div class="checker-board">

<OneChecker :index="1" :x="state.x[1]" :y="state.y[1]" />

<OneChecker :index="2" :x="state.x[2]" :y="state.y[2]" />

<OneChecker :index="3" :x="state.x[3]" :y="state.y[3]" />

<OneChecker :index="4" :x="state.x[4]" :y="state.y[4]" />

<OneChecker :index="5" :x="state.x[5]" :y="state.y[5]" />

<OneChecker :index="6" :x="state.x[6]" :y="state.y[6]" />

<OneChecker :index="7" :x="state.x[7]" :y="state.y[7]" />

<OneChecker :index="8" :x="state.x[8]" :y="state.y[8]" />

<OneChecker :index="9" :x="state.x[9]" :y="state.y[9]" />

<Transition name="fade">

<div v-if="showSolved" class="board-mask">

<p>Solved</p>

<p>Congratulations!</p>

</div>

</Transition>

<Transition name="fade">

<div v-if="showPlaySolution" class="board-mask" @click="handlePlaySolution">

<p>Click to play solution</p>

<p>Awesome algorithm!</p>

</div>

</Transition>

<Transition name="fade">

<div v-if="showNotSolvable" class="board-mask">

<p>Not Solvable</p>

<p>Try another one?</p>

</div>

</Transition>

</div>

</template>

<style lang="scss" scoped>

.checker-board {

position: relative;

width: 432px;

height: 432px;

background-color: #fff;

border-radius: 6px;

box-shadow: 2px 2px 3px rgb(0 0 0 / 20%);

}

.board-mask {

position: absolute;

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

width: 432px;

height: 432px;

font-size: 24px;

color: #333;

background-color: rgb(255 255 255 / 80%);

border-radius: 6px;

p:not(:last-child) {

margin-bottom: 12px;

font-size: 28px;

}

}

.fade-enter-active,

.fade-leave-active {

transition: opacity 0.5s ease;

}

.fade-enter-from,

.fade-leave-to {

opacity: 0;

}

</style>

<script lang="ts" setup>

import { useActions } from '@/scripts/actions';

import { useAlgorithm } from '@/scripts/algorithm';

import { useImage } from '@/scripts/image';

import { useMutables } from '@/scripts/mutables';

import { Direction, Stage, type GlobalState } from '@/types';

import { inject, onMounted, watch } from 'vue';

const state = inject('state') as GlobalState;

const { uploadBackground } = useImage();

const {

puzzleStringValidator,

analyzeSolvability,

move,

getRandomPuzzleString,

applyCurrentPuzzleString,

} = useActions();

const {

initialStateInputValidity,

targetStateInputValidity,

inputDisabled,

searchConsumption,

solutionSteps,

randomInitDisabled,

searchDisabled,

manualControlState,

manualControlDisabled,

luckyDogState,

luckyDogDisabled,

uploadBackgroundDisabled,

} = useMutables();

const { bfs, aStar } = useAlgorithm();

const handleInitialStateInput = () => {

if (puzzleStringValidator(state.initial)) {

initialStateInputValidity.value = true;

state.current = state.initial;

// If lucky dog selected, analyze the solvability

if (state.lucky) analyzeSolvability();

} else {

initialStateInputValidity.value = false;

}

};

const handleTargetStateInput = () => {

if (puzzleStringValidator(state.target)) {

targetStateInputValidity.value = true;

// If lucky dog selected, analyze the solvability

if (state.lucky) analyzeSolvability();

} else {

targetStateInputValidity.value = false;

}

};

const handleRandomInit = () => {

state.initial = getRandomPuzzleString();

state.current = state.initial;

// If lucky dog selected, analyze the solvability

if (state.lucky) analyzeSolvability();

};

const handleUninformedSearch = async () => {

// If not solvable, exit

analyzeSolvability();

if (state.stage !== Stage.Solvable) return;

state.stage = Stage.UninformedSearching;

// Wait 10ms for Vue.js state updating

setTimeout(() => {

const { solution, spaceConsumption, timeConsumption } = bfs();

state.spaceConsumption = spaceConsumption;

state.timeConsumption = timeConsumption;

state.steps = solution.length - 1;

solution.reverse();

state.solution = solution;

state.stage = Stage.UninformedSearchEnd;

}, 10);

};

const handleAStarSearch = async () => {

// If not solvable, exit

analyzeSolvability();

if (state.stage !== Stage.Solvable) return;

state.stage = Stage.AStarSearching;

// Wait 10ms for Vue.js state updating

setTimeout(() => {

const { solution, spaceConsumption, timeConsumption } = aStar();

state.spaceConsumption = spaceConsumption;

state.timeConsumption = timeConsumption;

state.steps = solution.length - 1;

solution.reverse();

state.solution = solution;

state.stage = Stage.AStarSearchEnd;

}, 10);

};

const handleManualControl = () => {

if (state.stage !== Stage.ManualControl) {

state.stage = Stage.ManualControl;

state.steps = 0;

} else {

state.stage = Stage.Solvable;

state.current = state.initial;

}

};

const handleLuckyDogSelect = () => {

state.stage = Stage.LuckyDogSelecting;

};

const handleUploadBackground = () => {

uploadBackground();

};

onMounted(() => {

// Automatically update the view when state.current is updated

watch(

() => state.current,

() => applyCurrentPuzzleString()

);

// Global keydown event listener

document.addEventListener('keydown', (e: KeyboardEvent) => {

if (state.stage === Stage.ManualControl) {

switch (e.key) {

case 'ArrowUp':

move(Direction.Up);

state.steps++;

break;

case 'ArrowDown':

move(Direction.Down);

state.steps++;

break;

case 'ArrowLeft':

move(Direction.Left);

state.steps++;

break;

case 'ArrowRight':

move(Direction.Right);

state.steps++;

break;

}

}

});

});

</script>

<template>

<div class="control-card">

<p>Consumption: {{ searchConsumption }}</p>

<p>Solution steps: {{ solutionSteps }}</p>

<div class="input-area">

<label>Initial state</label>

<input

v-model="state.initial"

type="text"

placeholder="Yes, it's necessary."

maxlength="9"

:disabled="inputDisabled"

@input="handleInitialStateInput"

@keydown.enter="handleInitialStateInput"

/>

<span>{{ initialStateInputValidity ? 'OK' : 'Bad' }}</span>

</div>

<div class="input-area">

<label>Target state</label>

<input

v-model="state.target"

type="text"

placeholder="Any idea?"

maxlength="9"

:disabled="inputDisabled"

@input="handleTargetStateInput"

@keydown.enter="handleTargetStateInput"

/>

<span>{{ targetStateInputValidity ? 'OK' : 'Bad' }}</span>

</div>

<button :disabled="randomInitDisabled" @click="handleRandomInit">Randomly initialize</button>

<button :disabled="searchDisabled" @click="handleUninformedSearch">

Uninformed search (BFS)

</button>

<button :disabled="searchDisabled" @click="handleAStarSearch">A\* search</button>

<button :disabled="manualControlDisabled" @click="handleManualControl">

Manual control: {{ manualControlState }}

</button>

<button :disabled="luckyDogDisabled" @click="handleLuckyDogSelect">

Lucky dog: {{ luckyDogState }}

</button>

<button :disabled="uploadBackgroundDisabled" @click="handleUploadBackground">

Upload background

</button>

</div>

</template>

<style lang="scss" scoped>

.control-card {

display: flex;

flex-direction: column;

justify-content: space-between;

width: 240px;

height: 350px;

padding: 12px;

font-size: 14px;

background-color: #fff;

border-radius: 6px;

box-shadow: 2px 2px 3px rgb(0 0 0 / 20%);

button {

color: inherit;

}

}

.input-area {

position: relative;

display: flex;

align-items: center;

height: 22px;

label {

width: 88px;

}

input {

box-sizing: border-box;

width: calc(100% - 88px);

color: inherit;

}

span {

position: absolute;

right: 6px;

font-size: 12px;

text-align: end;

}

}

</style>

<script lang="ts" setup>

import { useActions } from '@/scripts/actions';

import { Stage, type GlobalState } from '@/types';

import { computed, inject } from 'vue';

const props = defineProps({

index: {

type: Number,

required: true,

},

x: {

type: Number,

required: true,

},

y: {

type: Number,

required: true,

},

});

const state = inject('state') as GlobalState;

const { analyzeSolvability } = useActions();

const trySelect = () => {

if (state.stage === Stage.LuckyDogSelecting) {

state.lucky = props.index;

analyzeSolvability();

}

};

const backgroundSrc = computed(() => {

for (let i = 0; i < 9; i++) {

if (props.index.toString(10) === state.target[i]) {

return state.backgrounds[i];

}

}

return '';

});

</script>

<template>

<div

class="checker"

:style="{ left: `${x \* 134 + 16}px`, top: `${y \* 134 + 16}px` }"

@click="trySelect"

>

<img v-if="backgroundSrc" class="checker-background" :src="backgroundSrc" />

<div v-else class="checker-background" />

<div class="checker-index">

<span>{{ props.index }}</span>

</div>

</div>

</template>

<style lang="scss" scoped>

.checker {

position: absolute;

width: 128px;

height: 128px;

font-size: 3rem;

color: #fff;

border: 2px solid #000;

transition: all 0.5s;

.checker-background {

width: 100%;

height: 100%;

background-color: #fff;

}

.checker-index {

position: absolute;

top: 50%;

left: 50%;

display: flex;

align-items: center;

justify-content: center;

width: 100%;

height: 100%;

background-color: rgb(0 0 0 / 20%);

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

}

}

</style>

<script lang="ts" setup>

import AuthorCard from './AuthorCard.vue';

import ControlCard from './ControlCard.vue';

</script>

<template>

<div class="side-panel">

<ControlCard />

<AuthorCard />

</div>

</template>

<style lang="scss" scoped>

.side-panel {

margin-left: 12px;

}

</style>

import { Direction, Stage, type GlobalState } from '@/types';

import { inject } from 'vue';

export const useActions = () => {

const state = inject('state') as GlobalState;

/\*\*

\* Check if puzzle string is valid

\*/

const puzzleStringValidator = (str: string) => {

if (str.length !== 9) {

// Illegal puzzle string, length not equal to 9

return false;

} else {

for (let i = 1; i < 10; i++) {

if (str.indexOf(i.toString(10)) === -1) {

// Illegal puzzle string, can't find all numbers from 1 to 9

return false;

}

}

}

// Everything is OK

return true;

};

/\*\*

\* Check if the problem is solvable (Implemented by comparing number of inversions)

\*/

const isSolvable = () => {

/\*\* Simple inversions counting function \*/

const inversionsCount = (str: string) => {

let temp = str;

// Remove the lucky dog

temp = temp.replace(state.lucky.toString(10), '');

let sum = 0;

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

const partialString = temp.slice(i + 1);

for (const item of partialString) {

if (Number.parseInt(item) < Number.parseInt(temp[i])) {

sum++;

}

}

}

return sum;

};

if (inversionsCount(state.initial) % 2 === inversionsCount(state.target) % 2) {

// Solvable

return true;

} else {

// Not solvable

return false;

}

};

/\*\*

\* Check if the problem is solvable

\*/

const analyzeSolvability = () => {

if (isSolvable()) {

state.stage = Stage.Solvable;

} else {

state.stage = Stage.NotSolvable;

}

};

/\*\*

\* Swap the lucky dog with another. This will update state.current immediately.

\*

\* @param direction - 0: Up, 1: Down, 2: Left, 3: Right

\*/

const move = (direction: Direction) => {

/\*\* Simple swap function for state.current \*/

const swap = (a: number, b: number) => {

const arr = state.current.split('');

const temp = arr[a];

arr[a] = arr[b];

arr[b] = temp;

state.current = arr.join('');

};

const luckyIndex = state.current.indexOf(state.lucky.toString(10));

switch (direction) {

case Direction.Up:

if (luckyIndex > 2) swap(luckyIndex, luckyIndex - 3);

break;

case Direction.Down:

if (luckyIndex < 6) swap(luckyIndex, luckyIndex + 3);

break;

case Direction.Left:

if (luckyIndex % 3 > 0) swap(luckyIndex, luckyIndex - 1);

break;

case Direction.Right:

if (luckyIndex % 3 < 2) swap(luckyIndex, luckyIndex + 1);

break;

}

};

/\*\*

\* Randomly generate a puzzle string, but not guaranteed to have a solution.

\*/

const getRandomPuzzleString = () => {

let str = '';

const unused = ['1', '2', '3', '4', '5', '6', '7', '8', '9'];

while (unused.length > 1) {

str = str + unused.splice(Math.floor(Math.random() \* (unused.length + 1) - 1), 1)[0];

}

return str + unused[0];

};

/\*\*

\* Apply the current puzzle string to the checkerboard.

\*/

const applyCurrentPuzzleString = () => {

// Result schema ready

const result = {

x: [null, 0, 0, 0, 0, 0, 0, 0, 0, 0],

y: [null, 0, 0, 0, 0, 0, 0, 0, 0, 0],

} as {

x: [null, number, number, number, number, number, number, number, number, number];

y: [null, number, number, number, number, number, number, number, number, number];

};

// Calculate the coordinates of the corresponding checker based on the position where the number appears

for (let i = 0; i < 9; i++) {

result.x[Number.parseInt(state.current[i])] = i % 3;

result.y[Number.parseInt(state.current[i])] = Math.floor(i / 3);

}

// Assign result to state

Object.assign(state, result);

};

return {

puzzleStringValidator,

analyzeSolvability,

move,

getRandomPuzzleString,

applyCurrentPuzzleString,

};

};

import { Direction, type GlobalState } from '@/types';

import { inject } from 'vue';

/\*\* Simple swap function for string \*/

const swap = (str: string, a: number, b: number) => {

const arr = str.split('');

const temp = arr[a];

arr[a] = arr[b];

arr[b] = temp;

return arr.join('');

};

export const useAlgorithm = () => {

const state = inject('state') as GlobalState;

/\*\* Breadth-first search \*/

const bfs = () => {

// Get start time

const startTime = new Date();

const { initial, target } = state;

const lucky = state.lucky.toString(10);

const opened = new Array<string>();

const closed = new Map<string, string>();

const solution = new Array<string>();

// Push the initial state into the opened queue

opened.push(initial);

// Continuously expand the opened node and store results in the closed map

while (opened.length > 0) {

const current = opened.shift() as string;

// Push the current state into the closed map

// The first time through the loop, the second parameter here is undefined, but it doesn't matter

closed.set(current, closed.get(current) as string);

// Check if current node is target node

if (current === target) {

solution.push(current);

// Generate solution

let previous = closed.get(current);

while (previous) {

solution.push(previous);

previous = closed.get(previous);

}

break;

}

// Get index of the lucky dog

const indexOfLucky = current.indexOf(lucky);

// Generate all states that can reach in its next step

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

let temp;

if (direction === Direction.Up && indexOfLucky > 2) {

temp = swap(current, indexOfLucky, indexOfLucky - 3);

} else if (direction === Direction.Down && indexOfLucky < 6) {

temp = swap(current, indexOfLucky, indexOfLucky + 3);

} else if (direction === Direction.Left && indexOfLucky % 3 > 0) {

temp = swap(current, indexOfLucky, indexOfLucky - 1);

} else if (direction === Direction.Right && indexOfLucky % 3 < 2) {

temp = swap(current, indexOfLucky, indexOfLucky + 1);

}

// Unreachable, skip

if (!temp) continue;

// Check if the result already exists in the closed map,

// if not, push it into the opened queue

if (!closed.has(temp)) {

opened.push(temp);

closed.set(temp, current);

}

}

}

return {

solution,

spaceConsumption: opened.length + closed.size,

timeConsumption: (new Date().valueOf() - startTime.valueOf()) / 1000,

};

};

const aStar = () => {

/\*\* State Node \*/

class State {

/\*\* Current state value \*/

value: string;

/\*\* Previous state value \*/

parent: string;

/\*\* g(n): The cost of the path from the start node to n \*/

g: number;

/\*\* h(n): A heuristic function that estimates the cost of the cheapest path from n to the goal \*/

h: number;

/\*\* f(n) = g(n) + h(n) \*/

get f() {

return this.g + this.h;

}

constructor(value: string, parent: string, g: number) {

this.value = value;

this.parent = parent;

this.g = g;

// h(n): The sum of the distances from the current position of all checkers to the target position

// (Manhattan distance)

let sum = 0;

for (let currentIndex = 0; currentIndex < 9; currentIndex++) {

let targetIndex = 0;

// Before constructing, target must have been assigned a value

while (value[currentIndex] !== target[targetIndex]) {

targetIndex++;

}

// <index> % 3 is x-coordinate value

// <index> / 3 is y-coordinate value

sum +=

Math.abs(Math.floor(currentIndex % 3) - Math.floor(targetIndex % 3)) +

Math.abs(Math.floor(currentIndex / 3) - Math.floor(targetIndex / 3));

}

this.h = sum;

}

}

// Get start time

const startTime = new Date();

const { initial, target } = state;

const lucky = state.lucky.toString(10);

const opened = new Array<State>();

const closed = new Map<string, string>();

const solution = new Array<string>();

// Push the initial state into the opened queue

opened.push(new State(initial, '', 0));

// Continuously expand the best node

while (opened.length > 0) {

/\*\* Get the best node \*/

const getBestNode = () => {

// Find smallest f(n)

let minItemIndex = 0;

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

if (opened[i].f < opened[minItemIndex].f) {

minItemIndex = i;

}

}

// Returns the best node and removes it from the queue

// Priority queue is partially implemented

const temp = opened[minItemIndex];

opened.splice(minItemIndex, 1);

return temp;

};

const current = getBestNode();

// Push the current state into the closed map

// The first time through the loop, the second parameter here is undefined, but it doesn't matter

closed.set(current.value, current.parent);

// Check if current node is target node

if (current.value === target) {

solution.push(current.value);

// Generate solution

let previous = closed.get(current.value);

while (previous) {

solution.push(previous);

previous = closed.get(previous);

}

break;

}

// Get index of the lucky dog

const indexOfLucky = current.value.indexOf(lucky);

// Generate all states that can reach in its next step

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

let temp;

if (direction === Direction.Up && indexOfLucky > 2) {

temp = swap(current.value, indexOfLucky, indexOfLucky - 3);

} else if (direction === Direction.Down && indexOfLucky < 6) {

temp = swap(current.value, indexOfLucky, indexOfLucky + 3);

} else if (direction === Direction.Left && indexOfLucky % 3 > 0) {

temp = swap(current.value, indexOfLucky, indexOfLucky - 1);

} else if (direction === Direction.Right && indexOfLucky % 3 < 2) {

temp = swap(current.value, indexOfLucky, indexOfLucky + 1);

}

// Unreachable state, skip

if (!temp) continue;

// Check if the result already exists in the closed map,

// if not, check if it is a better node

if (!closed.has(temp)) {

let notFoundInOpened = true;

const tempState = new State(temp, current.value, current.g + 1);

// Check if the result already exists in opened queue

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

if (opened[i].value === temp) {

// Existed!

notFoundInOpened = false;

// Check if it is a better node

if (tempState.f < opened[i].f) {

// Yes, it is a better node! Update opened[i]

opened[i] = tempState;

}

// It is not a better node, nothing to do

break;

}

}

// Not found in opened queue

if (notFoundInOpened) {

// Insert

opened.push(tempState);

}

}

}

}

return {

solution,

spaceConsumption: opened.length + closed.size,

timeConsumption: (new Date().valueOf() - startTime.valueOf()) / 1000,

};

};

return { bfs, aStar };

};

import type { GlobalState } from '@/types';

import { inject } from 'vue';

export const useImage = () => {

const state = inject('state') as GlobalState;

/\*\* Calculate the coordinate value of the center clip \*/

const getCropInformation = (image: HTMLImageElement) => {

const width = image.naturalWidth;

const height = image.naturalHeight;

const min = width < height ? width : height;

return {

startX: (width - min) / 2,

startY: (height - min) / 2,

width: min,

height: min,

};

};

const setBackground = (e: Event) => {

// eslint-disable-next-line @typescript-eslint/ban-ts-comment

// @ts-ignore:next-line

const file = e.target.files[0] as File;

// Load image

const image = document.createElement('img');

image.src = URL.createObjectURL(file);

image.onload = () => {

// Create canvas for image (Center crop)

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

const context = canvas.getContext('2d') as CanvasRenderingContext2D;

const cropInformation = getCropInformation(image);

canvas.width = 384;

canvas.height = 384;

context.drawImage(

image,

cropInformation.startX,

cropInformation.startY,

cropInformation.width,

cropInformation.height,

0,

0,

384,

384

);

for (let x = 0; x < 3; x++) {

for (let y = 0; y < 3; y++) {

const newCanvas = document.createElement('canvas');

const newContext = newCanvas.getContext('2d') as CanvasRenderingContext2D;

newCanvas.width = 128;

newCanvas.height = 128;

newContext.drawImage(canvas, 128 \* x, 128 \* y, 128, 128, 0, 0, 128, 128);

state.backgrounds[y \* 3 + x] = newCanvas.toDataURL();

}

}

};

};

const uploadBackground = () => {

// Request image file input

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

input.type = 'file';

input.accept = 'image/\*';

input.onchange = setBackground;

input.click();

};

return { uploadBackground };

};

import { Stage, type GlobalState } from '@/types';

import { computed, inject, ref } from 'vue';

export const useMutables = () => {

const state = inject('state') as GlobalState;

const initialStateInputValidity = ref(true);

const targetStateInputValidity = ref(true);

const inputDisabled = computed(() => {

if (

state.stage === Stage.Initialized ||

state.stage === Stage.Solvable ||

state.stage === Stage.NotSolvable

) {

return false;

} else {

return true;

}

});

const searchConsumption = computed(() => {

if (

state.stage === Stage.UninformedSearchEnd ||

state.stage === Stage.AStarSearchEnd ||

state.stage === Stage.PlayingSolution

) {

return `${state.spaceConsumption.toString(10)} / ${state.timeConsumption.toFixed(1)}s`;

} else if (state.stage === Stage.UninformedSearching || state.stage === Stage.AStarSearching) {

return 'Searching...';

} else {

return 'N/A';

}

});

const solutionSteps = computed(() => {

if (

state.stage === Stage.UninformedSearchEnd ||

state.stage === Stage.AStarSearchEnd ||

state.stage === Stage.PlayingSolution ||

state.stage === Stage.ManualControl

) {

return state.steps === -1 ? 'N/A' : state.steps.toString(10);

} else if (state.stage === Stage.UninformedSearching || state.stage === Stage.AStarSearching) {

return 'Searching...';

} else {

return 'N/A';

}

});

const randomInitDisabled = computed(() => {

if (

state.stage === Stage.UninformedSearching ||

state.stage === Stage.AStarSearching ||

state.stage === Stage.PlayingSolution ||

state.stage === Stage.ManualControl

) {

return true;

} else {

return false;

}

});

const searchDisabled = computed(() => {

if (

state.stage === Stage.Solvable ||

state.stage === Stage.UninformedSearchEnd ||

state.stage === Stage.AStarSearchEnd

) {

return false;

} else {

return true;

}

});

const manualControlState = computed(() => {

if (state.stage === Stage.ManualControl) {

return 'Enable';

} else {

return 'Disable';

}

});

const manualControlDisabled = computed(() => {

if (state.stage === Stage.Solvable || state.stage === Stage.ManualControl) {

return false;

} else {

return true;

}

});

const luckyDogState = computed(() => {

if (state.stage === Stage.Initialized) {

return 'Wait for selection';

} else if (state.stage === Stage.LuckyDogSelecting) {

return 'Selecting...';

} else {

return `Selected ${state.lucky}`;

}

});

const luckyDogDisabled = computed(() => {

if (

state.stage === Stage.Initialized ||

state.stage === Stage.Solvable ||

state.stage === Stage.NotSolvable

) {

return false;

} else {

return true;

}

});

const uploadBackgroundDisabled = computed(() => {

if (

state.stage === Stage.Initialized ||

state.stage === Stage.Solvable ||

state.stage === Stage.NotSolvable

) {

return false;

} else {

return true;

}

});

return {

initialStateInputValidity,

targetStateInputValidity,

inputDisabled,

searchConsumption,

solutionSteps,

randomInitDisabled,

searchDisabled,

manualControlState,

manualControlDisabled,

luckyDogState,

luckyDogDisabled,

uploadBackgroundDisabled,

};

};

export enum Direction {

Up,

Down,

Left,

Right,

}

export enum Stage {

Initialized,

LuckyDogSelecting,

Solvable,

NotSolvable,

ManualControl,

UninformedSearching,

AStarSearching,

UninformedSearchEnd,

AStarSearchEnd,

PlayingSolution,

}

export type GlobalState = {

stage: Stage;

steps: number;

solution: string[];

spaceConsumption: number;

timeConsumption: number;

lucky: number;

initial: string;

target: string;

current: string;

x: [null, number, number, number, number, number, number, number, number, number];

y: [null, number, number, number, number, number, number, number, number, number];

backgrounds: [string, string, string, string, string, string, string, string, string];

};

<script lang="ts" setup>

import type { GlobalState } from '@/types';

import { provide, reactive } from 'vue';

import CheckerBoard from './components/CheckerBoard.vue';

import SidePanel from './components/SidePanel.vue';

// Global shared state

const state = reactive({

stage: 0,

steps: 0,

solution: [],

spaceConsumption: 0,

timeConsumption: 0,

lucky: 0,

initial: '123456789',

target: '123456789',

current: '123456789',

x: [null, 0, 1, 2, 0, 1, 2, 0, 1, 2],

y: [null, 0, 0, 0, 1, 1, 1, 2, 2, 2],

backgrounds: ['', '', '', '', '', '', '', '', ''],

}) as GlobalState;

provide('state', state);

</script>

<template>

<CheckerBoard />

<SidePanel />

<div class="viewport-error">

<p>Your viewport is too small</p>

<p>This application need 720px x 450px viewport at least</p>

</div>

</template>

<style lang="scss">

// Remove default padding and margin

\* {

padding: 0;

margin: 0;

}

// Body styles

body {

// Let the browser choose the appropriate sans-serif font. This feature has been supported since Chrome 56.

font-family: system-ui, sans-serif;

}

// Main container

#app {

display: flex;

align-items: center;

justify-content: center;

width: 100vw;

height: 100vh;

overflow: hidden;

color: #1b1b1b;

user-select: none;

background-color: #f2f2f2;

}

.viewport-error {

position: absolute;

top: 50%;

left: 50%;

box-sizing: border-box;

display: none;

flex-direction: column;

align-items: center;

justify-content: center;

width: 100vw;

height: 100vh;

padding: 12px;

font-size: 16px;

background-color: #f2f2f2;

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

p:first-child {

margin-bottom: 16px;

font-size: 22px;

font-weight: bold;

}

}

@media screen and (max-width: 720px) {

.viewport-error {

display: flex;

}

}

@media screen and (max-height: 450px) {

.viewport-error {

display: flex;

}

}

</style>

import { createApp } from 'vue';

import App from './App.vue';

const app = createApp(App);

app.mount('#app');

<!DOCTYPE html>

<html lang="en-US">

<head>

<meta charset="UTF-8" />

<link href="/favicon.ico" rel="icon" />

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

<title>8 Puzzle Problem</title>

</head>

<body>

<div id="app"></div>

<script src="/src/main.ts" type="module"></script>

</body>

</html>

## 实验结果测试与分析

图片略

判断有效性之后，两种算法都能搜索到正确的路径，而且A\*算法所花时间明显比bfs快

## 思考及学习心得

复习了bfs，学习了对于八数码这一特定问题的启发函数，以及启发式搜索，对于图形界面的实现，由于非常困难，借助了网上搜索的文档，最终还是顺利实现了