

# LAB5 马踏棋盘

实验题目：马踏棋盘

问题描述：设计一个国际象棋马踏遍棋盘的程序。将马随机放在国际象棋棋盘的某个方格上，按照马走棋规则移动，每个方格只进入一次，走遍棋盘上全部64个方格。

解题要求：求出马的行走路线，将数字1, 2, ..., 64依次填入8×8的方阵并输出。

## 1.需求分析

- 手动走棋：生成棋盘并手动挑战走棋，可悔棋
- 自动判断和走棋【核心】：实现算法来判断继续完成的解法数/完成剩余步骤
- 多棋盘：可同时开启多个棋盘挑战并切换
- 分享需求：通过下载棋盘信息并传递让好友来挑战残局

## 2.概要设计

### 2.1 设计思想

架构：前后端分离，后端主要负责算法和保存相关，前端负责图形化渲染和交互

算法核心：Warnsdorff贪心策略，优先选择后续可走路线最少的节点进行深搜和回溯

### 2.2 存储结构设计

棋盘存储结构采取两种方式，根据前后端交互任务的性质而采取不同的策略

**走棋**相关：

需求：尽可能地快返回结果，因此传输完整棋盘来“空间换时间”，避免解析时间。

设计：前端->后端 当前棋盘大小 n / 状态 n\*n (-1为已落子，0为空) / 当前落子位置

前端<-后端 路径栈

**棋局保存**相关：

需求：需要保存在后端数据库，因此要尽可能节省空间，同时要保存路径信息。故采取路径栈来“时间换空间”。

设计：前端<->后端 当前棋盘大小 n / 路径栈

此外，棋盘参数使用类的成员存储，避免深搜时大量递归调用爆栈

## 3.详细设计

### 后端

**实现：**python的flask框架 + SQLite数据库

**接口设计与功能：**

- 走棋相关：表单json传参当前棋盘

`/api/auto-check` 判断继续走棋的解法

`/api/auto-move` 自动完成剩余走棋步骤，分段返回剩下k步路径（每步通过1-8来标识跳法）

- 棋局保存相关：表单json传参 当前棋盘大小 n 以及 已走路径（每步通过1-8来标识跳法），存储入 SQLite (uuid为key)

`/api/save` 保存残局棋盘到指定保存位 (1/2/3)

`/api/recall` 从指定保存位恢复棋盘

`/api/download` 下载当前残局棋盘，返回json文件

`/api/import` 上传棋盘json文件到当前棋盘

## 前端

实现：html + JavaScript + CSS 三件套

详细功能：

- 棋盘生成：根据设定 n(4-10) 值生成  $n \times n$  棋盘，并随机初始化马的初始位置
- 判断当前棋盘继续走棋可能性：直接调用接口 `/api/auto-check`
- 继续走棋：调用接口 `/api/auto-move`，并动画演示走棋结果（将步数数字填入棋盘）
- 手动挑战：鼠标点击下一步位置，js判断落子是否合法
- 悔棋：通过前端js维护一个路径栈实现，该栈在棋局保存时会被传给后端，在恢复/上传时会被同步更新
- 棋局保存相关：调用对应api，渲染棋局。需要注意提醒用户若恢复/上传会清空当前棋盘

## 4. 实验结果

截图如下：



## 5. 附录（程序代码）

## app.py 负责创建实例、接口处理和数据库维护

```

from flask import Flask, request, jsonify, send_file, render_template, make_response
import json
import sqlite3
import uuid
import os
from chess import Chesssolver

app = Flask(__name__)

def init_db(): # 初始化数据库
    conn = sqlite3.connect('chess.db')
    c = conn.cursor()
    c.execute('''
        CREATE TABLE IF NOT EXISTS saved_games
        (id INTEGER PRIMARY KEY AUTOINCREMENT,
        uid TEXT,
        n INTEGER,
        path TEXT,
        initX INTEGER,
        initY INTEGER,
        save_loc INTEGER)
    ''')
    conn.commit()
    conn.close()

init_db()

@app.route('/')
def index(): # 主页面
    uid = request.cookies.get("user")
    if not uid:
        uid = str(uuid.uuid4())
    response = make_response(render_template('index.html'))
    response.set_cookie("user", uid)
    return response

@app.route('/api/auto-check', methods=['POST'])
def auto_check(): # 检查当前棋盘是否可以继续完成
    data = request.json
    n = data.get('n')
    board = data.get('board')
    current_pos = data.get('current_pos')

    solver = Chesssolver(n, board, current_pos)
    solver.count()
    ans = solver.counter

    return jsonify({
        'check': ans != 0,
        'message': f'共有{ans}种解法' if ans <= 200 else f'还有至少200种解法'
    })

@app.route('/api/auto-move', methods=['POST'])
def auto_move(): # 自动完成剩余走棋步骤

```

```

data = request.json
n = data.get('n')
board = data.get('board')
current_pos = data.get('current_pos')

solver = ChessSolver(n, board, current_pos)
success = solver.solve()
return jsonify({
    'success': success,
    'path': solver.path
})

@app.route('/api/save', methods=['POST'])
def save_game():
    uid = request.cookies.get("user")
    if not uid:
        return jsonify({
            'success': False,
            'message': '缺少用户标识数据'
        })

    data = request.json
    n = data.get('n')
    path = data.get('path')
    init_pos = data.get('init_pos')
    save_loc = data.get('save_loc')

    conn = sqlite3.connect('chess.db')
    c = conn.cursor()

    # 检查是否已存在该保存位
    c.execute('SELECT * FROM saved_games WHERE uid = ? AND save_loc = ?', (uid, save_loc))
    existing = c.fetchone()

    if existing:
        c.execute('UPDATE saved_games SET n=? , path=? , initX=? , initY=? WHERE
        uid=? AND save_loc=?',
                  (n, json.dumps(path), init_pos[0], init_pos[1], uid, save_loc))
    else:
        c.execute('INSERT INTO saved_games (uid, n, path, initX, initY,
        save_loc) VALUES (?, ?, ?, ?, ?, ?, ?)',
                  (uid, n, json.dumps(path), init_pos[0], init_pos[1], save_loc))

    conn.commit()
    conn.close()

    return jsonify({
        'success': True,
        'message': f'棋局已保存到位置{save_loc}'
    })

@app.route('/api/recall', methods=['GET'])
def recall_game():
    uid = request.cookies.get("user")
    if not uid:
        return jsonify({
            'success': False,
            'message': '缺少用户标识数据'
        })

```

```

'message': '缺少用户标识数据'
})

save_loc = request.args.get('save_loc', type=int)

conn = sqlite3.connect('chess.db')
c = conn.cursor()
c.execute('SELECT n, path, initX, initY FROM saved_games WHERE uid = ? AND
save_loc = ?', (uid, save_loc))
result = c.fetchone()
conn.close()

if result:
    n, path_json, initX, initY = result
    return jsonify({
        'success': True,
        'n': n,
        'path': json.loads(path_json),
        'init_pos': [initX, initY]
    })
else:
    return jsonify({
        'success': False,
        'message': f'保存位{save_loc}没有保存的棋局'
    })

@app.route('/api/download', methods=['POST'])
def download_game(): # 下载当前残局棋盘为JSON文件
    uid = request.cookies.get("user")
    if not uid:
        return jsonify({
            'success': False,
            'message': '缺少用户标识数据'
        })

    data = request.json

    filename = f"chess_game_{uid}.json"
    filepath = os.path.join('downloads', filename)

    os.makedirs('downloads', exist_ok=True)

    with open(filepath, 'w', encoding='utf-8') as f:
        json.dump(request.json, f, ensure_ascii=False, indent=2)

    return send_file(filepath, as_attachment=True, download_name=f"马踏棋盘残
局.json")

@app.route('/api/import', methods=['POST'])
def import_game(): # 上传棋盘JSON文件
    if 'file' not in request.files:
        return jsonify({'success': False, 'message': '没有上传文件'})

    file = request.files['file']
    if file.filename == '':
        return jsonify({'success': False, 'message': '没有选择文件'})

    try:

```

```

data = json.load(file)

return jsonify({
    'success': True,
    'n': data.get('n'),
    'path': data.get('path'),
    'init_pos': data.get('init_pos'),
    'message': '棋局导入成功'
})
except Exception as e:
    return jsonify({'success': False, 'message': f'文件解析失败: {str(e)}'})

if __name__ == '__main__':
    app.run(debug=True, port=5000)

```

## chess.py 负责计算解数、提供解

```

class chessSolver:
    def __init__(self, n, board, current_pos):
        self.n = n
        self.board = [row.copy() for row in board] # 深拷贝
        self.current_pos = current_pos
        self.moves = [
            (2, 1), (1, 2), (-1, 2), (-2, 1),
            (-2, -1), (-1, -2), (1, -2), (2, -1)
        ] # 八种跳法
        self.path = [] # 步数栈
        self.steps = self.n * self.n - sum(1 for row in self.board for _ in row)
    if _ != 0: # 还需要完成的步数
        self.counter = 0 # 统计解个数

    def go_back(self):
        if len(self.path) == 0:
            return False
        self.board[self.current_pos[0]][self.current_pos[1]] = 0
        idx = self.path.pop()
        self.current_pos = (self.current_pos[0] - self.moves[idx][0],
                            self.current_pos[1] - self.moves[idx][1])
        self.steps += 1
        return True

    def go_ahead(self, idx):
        self.path.append(idx)
        self.current_pos = (self.current_pos[0] + self.moves[idx][0],
                            self.current_pos[1] + self.moves[idx][1])
        self.board[self.current_pos[0]][self.current_pos[1]] = 1
        self.steps -= 1

    def within_board_check(self, x, y):
        return 0 <= x < self.n and 0 <= y < self.n

    def next_moves(self): # Warnsdorff贪心策略，优先选择后续可走路线最少的节点
        moves = []
        x, y = self.current_pos
        for i, (dx, dy) in enumerate(self.moves):
            nx, ny = x + dx, y + dy
            if self.within_board_check(nx, ny) and self.board[nx][ny] == 0:

```

```

ways = 0
for dx2, dy2 in self.moves:
    if self.within_board_check(nx + dx2, ny + dy2) and
self.board[nx + dx2][ny + dy2] == 0:
        ways += 1
moves.append((i, ways))
return sorted(moves, key=lambda x:x[-1]) # 按可走路线数量升序返回

def solve(self): # 解决剩余路径 DFS
if self.steps == 0:
    return True

for move_idx, _ in self.next_moves():
    self.go_ahead(move_idx)
    if self.solve(): return True
    self.go_back()

return False

def count(self): # 统计剩余解个数 DFS
if self.steps == 0:
    self.counter += 1
    if self.counter > 200:
        return True # 快速解决所有堆栈

for move_idx, _ in self.next_moves():
    self.go_ahead(move_idx)
    if self.count(): return True
    self.go_back()

return False

```

## static/script.js 负责处理用户交互和渲染

```

class ChessGame {
constructor() {
    this.n = 8;
    this.board = [];
    this.path = [];
    this.currentPos = null;
    this.stepCount = 0;

    this.isAutoMoving = false;
    this.remainingMoves = [];
    this.animationPromise = null;

    this.directions = [
        [2, 1], [1, 2], [-1, 2], [-2, 1],
        [-2, -1], [-1, -2], [1, -2], [2, -1]
    ];

    this.initEventListeners();
    this.generateBoard();
}

initEventListeners() {
    // 生成棋盘
}

```

```

document.getElementById('generateBoard').addEventListener('click', () =>
{
    this.generateBoard();
});

// 重置棋盘
document.getElementById('resetBoard').addEventListener('click', () => {
    this.resetBoard();
});

// 检查可行性
document.getElementById('autoCheck').addEventListener('click', () => {
    this.autoCheck();
});

// 自动走棋
document.getElementById('autoMove').addEventListener('click', () => {
    this.autoMove();
});

// 暂停自动走棋
document.getElementById('pauseAutoMove').addEventListener('click', () =>
{
    this.pauseAutoMove();
});

// 悔棋
document.getElementById('undoMove').addEventListener('click', () => {
    this.undoMove();
});

// 保存和恢复
document.getElementById('saveBtn').addEventListener('click', () => {
    this.saveGame(document.getElementById('save-loc').value);
});
document.getElementById('recallBtn').addEventListener('click', () => {
    this.recallGame(document.getElementById('recall-loc').value);
});

// 文件操作
document.getElementById('downloadBtn').addEventListener('click', () => {
    this.downloadGame();
});

document.getElementById('importBtn').addEventListener('click', () => {
    document.getElementById('importFile').click();
});

document.getElementById('importFile').addEventListener('change', (e) =>
{
    this.importGame(e.target.files[0]);
});

resetBoard() {
    this.board = Array(this.n).fill().map(() => Array(this.n).fill(0));
    this.path = [];
    this.stepCount = 0;
}

```

```

// 随机选择起始位置
const startX = Math.floor(Math.random() * this.n);
const startY = Math.floor(Math.random() * this.n);
this.initPos = [startX, startY]
this.currentPos = [startX, startY];
this.board[startX][startY] = 1;
this.stepCount = 1;

this.renderBoard();
document.getElementById('stepCount').textContent = this.stepCount;
document.getElementById('totalsteps').textContent = this.n * this.n;
this.showMessage(`棋盘已生成 ${this.n}x${this.n}, 马在位置 (${startX}, ${startY})`);
}

generateBoard() {
    this.n = parseInt(document.getElementById('boardsize').value);
    this.resetBoard();
}

renderBoard() {
    const chessboard = document.getElementById('chessboard');
    chessboard.innerHTML = '';
    chessboard.style.gridTemplateColumns = `repeat(${this.n}, 1fr)`;

    for (let i = 0; i < this.n; i++) {
        for (let j = 0; j < this.n; j++) {
            const cell = document.createElement('div');
            cell.className = `cell ${((i + j) % 2 === 0 ? 'light' : 'dark')}`;
            cell.dataset.x = i;
            cell.dataset.y = j;

            if (i === this.currentPos[0] && j === this.currentPos[1]) {
                cell.classList.add('current');
                cell.textContent = '马';
            } else if (this.board[i][j] !== 0) {
                cell.textContent = this.board[i][j];
                cell.classList.add('visited');
            }

            if (this.isPossibleMove(i, j)) {
                cell.classList.add('possible');
                cell.textContent = '$';
            }

            cell.addEventListener('click', () => this.handleCellClick(i, j));
            chessboard.appendChild(cell);
        }
    }

    document.getElementById('totalsteps').textContent = this.n * this.n;
    document.getElementById('stepCount').textContent = this.stepCount;
}

isPossibleMove(x, y) {
}

```

```

if (this.board[x][y] !== 0) return false;

const [currentX, currentY] = this.currentPos;
const dx = Math.abs(x - currentX);
const dy = Math.abs(y - currentY);

return (dx === 2 && dy === 1) || (dx === 1 && dy === 2);
}

handleCellClick(x, y) {
  if (this.isPossibleMove(x, y)) {
    this.makeMove(x, y);
  } else if (this.board[x][y] === 0) {
    this.showMessage('移动不合法！');
  } else {
    this.showMessage('目标位置已步过！');
  }
}

makeMove(x, y) {
  this.stepCount++;
  this.board[x][y] = this.stepCount;

  const [currentX, currentY] = this.currentPos;
  const dx = x - currentX;
  const dy = y - currentY;

  const direction = this.directions.findIndex(([dirX, dirY]) => dirX ===
dx && dirY === dy) + 1;
  this.path.push(direction); // 更新路径栈

  this.currentPos = [x, y];
  this.renderBoard();
  document.getElementById('stepCount').textContent = this.stepCount;

  if (this.stepCount === this.n * this.n) {
    this.showMessage('恭喜！你成功完成了马踏棋盘！');
  }
}

undoMove() {
  if (this.path.length === 0) {
    this.showMessage('无法悔棋！');
    return;
  }

  const [currentX, currentY] = this.currentPos;
  this.board[currentX][currentY] = 0;
  const lastDirection = this.path.pop();
  this.stepCount--;

  const [dx, dy] = this.directions[lastDirection - 1];
  this.currentPos = [currentX - dx, currentY - dy];

  this.renderBoard();
  document.getElementById('stepCount').textContent = this.stepCount;
  this.showMessage('悔棋一步');
}
}

```

```

async autoCheck() {
    try {
        const response = await fetch('/api/auto-check', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                n: this.n,
                board: this.board,
                current_pos: this.currentPos
            })
        });

        const data = await response.json();
        this.showMessage(data.message);
    } catch (error) {
        this.showMessage('检查失败: ' + error.message);
    }
}

pauseAutoMove() {
    if (this.isAutoMoving) {
        this.isAutoMoving = false;

        if (this.animationPromise) {
            this.animationPromise.cancel = true;
        }

        this.remainingMoves = [];
    }

    this.showMessage('自动走棋已暂停，未走步骤已清除');
} else {
    this.showMessage('当前没有进行中的自动走棋');
}
}

async autoMove() {
    if (this.isAutoMoving) {
        this.showMessage('自动走棋正在进行中');
        return;
    }

    try {
        const response = await fetch('/api/auto-move', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                n: this.n,
                board: this.board,
                current_pos: this.currentPos
            })
        });

        const data = await response.json();
    }
}

```

```

        if (data.success && data.path && data.path.length > 0) {
            this.isAutoMoving = true;
            this.remainingMoves = [...data.path];
            await this.animateMoves(data.path);

            if (this.isAutoMoving) {
                this.isAutoMoving = false;
                this.remainingMoves = [];
                this.showMessage('自动走棋完成！');
            }
        } else {
            this.showMessage('无法自动走棋');
        }
    } catch (error) {
        this.isAutoMoving = false;
        this.remainingMoves = [];
        this.showMessage('自动走棋失败: ' + error.message);
    }
}

async animateMoves(moves) {
    for (const [index, move] of moves.entries()) {
        if (!this.isAutoMoving) {
            this.remainingMoves = moves.slice(index);
            break;
        }

        this.animationPromise = new Promise((resolve) => {
            const timer = setTimeout(resolve, 1000);
            if (this.animationPromise) {
                this.animationPromise.cancel = () => {
                    clearTimeout(timer);
                    resolve();
                };
            }
        });
    });

    await this.animationPromise;

    if (!this.isAutoMoving) {
        this.remainingMoves = moves.slice(index);
        break;
    }

    const [dx, dy] = this.directions[move];
    const [currentX, currentY] = this.currentPos;
    const newX = currentX + dx;
    const newY = currentY + dy;

    this.makeMove(newX, newY);

    this.remainingMoves = moves.slice(index + 1);
}

this.animationPromise = null;
}

```

```

async saveGame(loc) {
    try {
        const response = await fetch('/api/save', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                n: this.n,
                path: this.path,
                init_pos: this.initPos,
                save_loc: parseInt(loc)
            })
        });
    }

    const data = await response.json();
    this.showMessage(data.message);
} catch (error) {
    this.showMessage('保存失败: ' + error.message);
}
}

async recallGame(loc) {
    if (!confirm('恢复棋局将清空当前棋盘，是否继续?')) return;

    try {
        const response = await fetch(`/api/recall?save_loc=${loc}`);
        const data = await response.json();

        if (data.success) {
            this.n = data.n;
            this.path = data.path;
            this.initPos = data.init_pos
            this.reconstructBoard();
            this.showMessage(`已恢复保存位${loc}的棋局`);
        } else {
            this.showMessage(data.message);
        }
    } catch (error) {
        this.showMessage('恢复失败: ' + error.message);
    }
}

reconstructBoard() {
    // 重新构建棋盘状态
    this.board = Array(this.n).fill().map(() => Array(this.n).fill(0));
    this.stepCount = 0;

    let currentX = this.initPos[0], currentY = this.initPos[1];
    this.board[currentX][currentY] = ++this.stepCount;

    // 按照路径栈重新走棋
    for (const move of this.path) {
        const [dx, dy] = this.directions[move - 1];
        currentX += dx;
        currentY += dy;
        this.board[currentX][currentY] = ++this.stepCount;
    }
}

```

```

this.currentPos = [currentX, currentY];
this.renderBoard();
document.getElementById('stepCount').textContent = this.stepCount;
}

async downloadGame() {
try {
const response = await fetch('/api/download', {
method: 'POST',
headers: {
'Content-Type': 'application/json',
},
body: JSON.stringify({
n: this.n,
path: this.path,
init_pos: this.initPos
})
});
}

const blob = await response.blob();
const url = window.URL.createObjectURL(blob);
const a = document.createElement('a');
a.href = url;
a.download = `马踏棋盘_${this.n}x${this.n}.json`;
document.body.appendChild(a);
a.click();
document.body.removeChild(a);
window.URL.revokeObjectURL(url);

this.showMessage('棋局文件已下载');
} catch (error) {
this.showMessage('下载失败: ' + error.message);
}
}

async importGame(file) {
if (!file) return;

if (!confirm('导入棋局将清空当前棋盘, 是否继续?')) return;

try {
const formData = new FormData();
formData.append('file', file);

const response = await fetch('/api/import', {
method: 'POST',
body: formData
});

const data = await response.json();

if (data.success) {
this.n = data.n;
this.path = data.path;
this.initPos = data.init_pos;
this.reconstructBoard();
this.showMessage('棋局导入成功');
}
}
}

```

```

        } else {
            this.showMessage(data.message);
        }
    } catch (error) {
    this.showMessage('导入失败: ' + error.message);
}
}

showMessage(message) {
const messageElement = document.getElementById('message');
messageElement.textContent = message;

setTimeout(() => {
if (messageElement.textContent === message) {
    messageElement.textContent = '';
}
}, 5000);
}
}

// 初始化游戏
document.addEventListener('DOMContentLoaded', () => {
new ChessGame();
});
}

```

## templates/index.html 负责页面元素布局

```

<!DOCTYPE html>
<html lang="zh-CN">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>马踏棋盘</title>
<link rel="stylesheet" href="static/style.css">
<script src="static/script.js"></script>
</head>
<body>
<div class="container">
<header>
<h1>马踏棋盘</h1>
<p>规则: 将马随机放在国际象棋棋盘的某个方格上, 按照马走棋规则移动, 每个方格只进入一次, 走遍棋盘上全部方格。</p>
</header>

<!-- 状态面板移到标题下方 -->
<div class="status-panel">
<div id="message" class="message">欢迎使用马踏棋盘! 请生成棋盘开始游戏。
</div>
<div class="info">
<span>当前步数: <span id="stepCount">0</span></span>
<span>总步数: <span id="totalSteps">64</span></span>
</div>
</div>

<div class="main-content">
<!-- 左侧控制面板 -->

```

```

<div class="left-panel">
    <div class="control-panel">
        <div class="config-section">
            <div class="config-row">
                <label for="boardsize">棋盘大小:</label>
                <select id="boardsize">
                    <option value="4"> 4 x 4</option>
                    <option value="5"> 5 x 5</option>
                    <option value="6"> 6 x 6</option>
                    <option value="7"> 7 x 7</option>
                    <option value="8" selected> 8 x 8</option>
                    <option value="9"> 9 x 9</option>
                    <option value="10">10 x 10</option>
                </select>
            </div>
            <div class="config-row">
                <button id="generateBoard">生成棋盘</button>
            </div>
            <div class="config-row">
                <button id="resetBoard">重置棋盘</button>
            </div>
            <div class="config-row">
                <button id="undoMove">悔棋</button>
            </div>
        </div>

        <div class="auto-section">
            <div class="auto-row">
                <button id="autoCheck">自动统计剩余解数</button>
            </div>
            <div class="auto-row">
                <button id="autoMove">自动走棋</button>
            </div>
            <div class="auto-row">
                <button id="pauseAutoMove">暂停自动走棋</button>
            </div>
        </div>
    </div>
</div>

<!-- 中央棋盘 -->
<div class="chessboard-container">
    <div id="chessboard" class="chessboard"></div>
</div>

<!-- 右侧保存面板 -->
<div class="right-panel">
    <div class="save-section">
        <div class="save-row">
            <label for="save-loc">保存至.....</label>
            <select id="save-loc">
                <option value="1"> 位置 1</option>
                <option value="2"> 位置 2</option>
                <option value="3"> 位置 3</option>
            </select>
            <button id="saveBtn">确认</button>
        </div>
    </div>

```

```

<div class="save-row">
    <label for="recall-loc">从.....恢复</label>
    <select id="recall-loc">
        <option value="1">位置 1</option>
        <option value="2">位置 2</option>
        <option value="3">位置 3</option>
    </select>
    <button id="recallBtn">确认</button>
</div>

<div class="save-row">
    <button id="downloadBtn">下载残局至本地</button>
</div>

<div class="save-row">
    <input type="file" id="importFile" accept=".json"
style="display: none;">
        <button id="importBtn">从本地上传残局</button>
    </div>
</div>
</div>
</body>
</html>

```

## static/style.css 负责页面元素美化

```

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Microsoft YaHei', Arial, sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    min-height: 100vh;
    padding: 20px;
}

.container {
    max-width: 1200px;
    margin: 0 auto;
    background: rgba(255, 255, 255, 0.95);
    border-radius: 15px;
    padding: 30px;
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);
}

header {
    text-align: center;
    margin-bottom: 20px;
}

header h1 {

```

```
color: #333;
font-size: 2.5em;
margin-bottom: 10px;
text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.1);
}

header p {
    color: #666;
    font-size: 1.1em;
}

/* 状态面板 */
.status-panel {
    background: #e9ecef;
    padding: 15px;
    border-radius: 8px;
    margin-bottom: 25px;
    text-align: center;
}

.message {
    font-size: 1.1em;
    font-weight: bold;
    color: #495057;
    margin-bottom: 10px;
    min-height: 24px;
}

.info {
    display: flex;
    justify-content: center;
    gap: 30px;
    font-size: 1em;
    color: #6c757d;
}

/* 主内容区域 */
.main-content {
    display: flex;
    justify-content: space-between;
    align-items: flex-start;
    gap: 20px;
}

/* 左侧面板 */
.left-panel {
    flex: 0 0 250px;
}

.control-panel {
    display: flex;
    flex-direction: column;
    gap: 20px;
}

.config-section, .auto-section {
    background: #f8f9fa;
    padding: 15px;
```

```

    border-radius: 8px;
    border: 1px solid #e9ecef;
}

.config-row, .auto-row {
    margin-bottom: 10px;
}

.config-row:last-child, .auto-row:last-child {
    margin-bottom: 0;
}

/* 棋盘容器 */
.chessboard-container {
    flex: 1;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 500px;
}

.chessboard {
    display: inline-grid;
    gap: 1px;
    background: #8b4513;
    padding: 2px;
    border: 3px solid #5d4037;
    box-shadow: 0 8px 25px rgba(0, 0, 0, 0.3);
}

/* 右侧面板 */
.right-panel {
    flex: 0 0 250px;
}

.save-section {
    background: #f8f9fa;
    padding: 15px;
    border-radius: 8px;
    border: 1px solid #e9ecef;
    display: flex;
    flex-direction: column;
    gap: 15px;
}

.save-row {
    display: flex;
    flex-direction: column;
    gap: 8px;
}

/* 通用样式 */
label {
    font-weight: bold;
    color: #495057;
    margin-bottom: 5px;
}

```

```

select, button {
    padding: 10px 15px;
    border: 2px solid #dee2e6;
    border-radius: 8px;
    font-size: 14px;
    transition: all 0.3s ease;
}

select {
    background: white;
    cursor: pointer;
}

button {
    background: #007bff;
    color: white;
    border: none;
    cursor: pointer;
    font-weight: bold;
    width: 100%;
}

button:hover {
    background: #0056b3;
    transform: translateY(-2px);
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
}

button:active {
    transform: translateY(0);
}

/* 棋盘格子样式 */
.cell {
    width: 50px;
    height: 50px;
    display: flex;
    align-items: center;
    justify-content: center;
    font-weight: bold;
    font-size: 14px;
    cursor: pointer;
    transition: all 0.3s ease;
    position: relative;
}

.cell.light {
    background: #f0d9b5;
}

.cell.dark {
    background: #b58863;
}

.cell:hover {
    transform: scale(1.05);
    z-index: 1;
    box-shadow: 0 0 10px rgba(255, 215, 0, 0.8);
}

```

```

}

.cell.current {
    background: #ffeb3b !important;
    color: #000;
    font-weight: bold;
}

.cell.visited {
    background: #4caf50 !important;
    color: white;
}

.cell.possible {
    background: #2196f3 !important;
    color: white;
    cursor: pointer;
}

.cell.possible:hover {
    background: #1976d2 !important;
}

/* 响应式设计 */
@media (max-width: 768px) {
    .container {
        padding: 15px;
    }

    .main-content {
        flex-direction: column;
    }

    .left-panel, .right-panel {
        flex: 1;
        width: 100%;
    }

    .cell {
        width: 35px;
        height: 35px;
        font-size: 12px;
    }

    header h1 {
        font-size: 2em;
    }

    .save-row {
        flex-direction: row;
        align-items: center;
    }

    .save-row label {
        margin-bottom: 0;
        margin-right: 10px;
    }
}

```

