DRL HW5 說明報告

1. 作業要求

(1) 目標

- 使用 RooCode 實現一遍 HW2 (Gridworld problem)
- 使用 Websim 做前端整合

(2) 預期結果

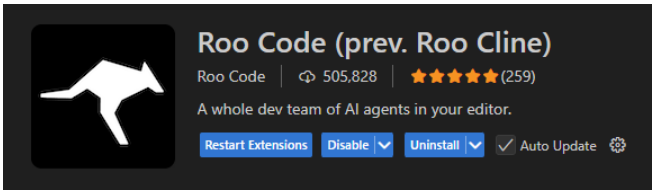
- 使用價值迭代算法 (Value Iteration Algorithm) 計算 Gridworld 環境的最佳政策 (Optimal Policy)。
- 在給定起點 (Start)、終點 (Goal) 和障礙物 (Obstacles) 的條件下，推導最佳行動策略。
- 計算價值函數 (Value Function V(s))，表示在最佳政策下，每個狀態的期望回報。
- 透過動畫模擬學習過程，並在動畫結束後將最優路徑標記成 黃色。
- 最佳政策顯示：
  - 每個格子顯示最佳行動 (↑、↓、←、→)，代表最優移動方向。
- 價值函數顯示：
  - 每個格子顯示 價值函數 V(s)，表示該狀態的最優回報。
- 動畫模擬路徑：
  - 根據策略矩陣，模擬學習過程。
  - 動畫結束後，將最終最佳路徑標記為黃色。

2. Roo Code 使用

(1) RooCode 是什麼？

RooCode 是一款 Visual Studio Code 的擴充套件，核心運作原理是透過串接像 ChatGPT、Gemini 等大型語言模型的 API key，讓我們能直接在 Visual Studio Code 中向這些 AI 工具發問，而無需額外開啟瀏覽器。同時，它也支援直接產生與編輯本地端的程式碼，讓整體開發流程更加流暢高效。

在這次 HW2 的 Gridworld 作業中，我使用 **Gemini 的 API key** 作為 RooCode 的 AI 協作來源，搭配本地的 Flask API 與前端畫面整合，幾乎全程都在 VSCode 環境中完成。這樣的開發方式不僅節省了大量時間，也讓我能更專注在演算法設計與實作上。



(2) RooCode Prompt

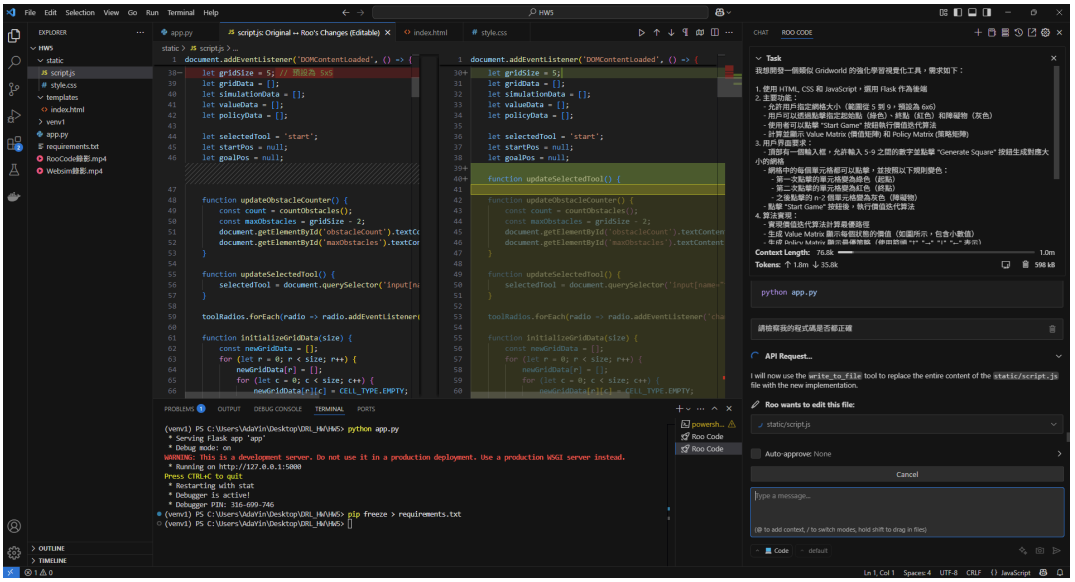
我想開發一個類似 Gridworld 的強化學習視覺化工具，需求如下：

1. 使用 HTML, CSS 和 JavaScript，選用 Flask 作為後端
2. 主要功能：
    - 允許用戶指定網格大小（範圍從 5 到 9，預設為 6x6）
    - 用戶可以透過點擊指定起始點（綠色）、終點（紅色）和障礙物（灰色）
    - 使用者可以點擊 "Start Game" 按鈕執行價值迭代算法
    - 計算並顯示 Value Matrix (價值矩陣) 和 Policy Matrix (策略矩陣)
3. 用戶界面要求：
    - 頂部有一個輸入框，允許輸入 5-9 之間的數字並點擊 "Generate Square" 按鈕生成對應大小的網格
    - 網格中的每個單元格都可以點擊，並按照以下規則變色：
        - 第一次點擊的單元格變為綠色（起點）
        - 第二次點擊的單元格變為紅色（終點）
        - 之後點擊的 n-2 個單元格變為灰色（障礙物）
    - 點擊 "Start Game" 按鈕後，執行價值迭代算法
4. 算法實現：
    - 實現價值迭代算法計算最優路徑
    - 生成 Value Matrix 顯示每個狀態的價值（如圖所示，包含小數值）
    - 生成 Policy Matrix 顯示最優策略（使用箭頭 "↑", "→", "↓", "←" 表示）
5. 模擬功能：
    - 添加一個模擬區域，顯示與主網格相同大小的網格
    - 在模擬區域中可視化強化學習代理的行為路徑
    - 代理從起點開始，按照最優策略（Policy Matrix）移動到終點
6. 視覺設計：
    - 整體佈局如圖所示
    - 使用淺色背景和簡潔的界面
    - Value Matrix 和 Policy Matrix 以表格形式展示
    - 網格使用適當的邊框和顏色區分不同類型的單元格

請提供完整的實現程式碼，確保所有功能正常運作。

(3) 實際使用 RooCode 截圖

如下方截圖所示，我在右側的 RooCode 指令輸入框中輸入需求，請它協助生成程式碼。它能直接在我本地的程式碼中逐行檢視並進行修改，讓整體開發流程更流暢，大幅提升了開發效率。



3. Websim 使用

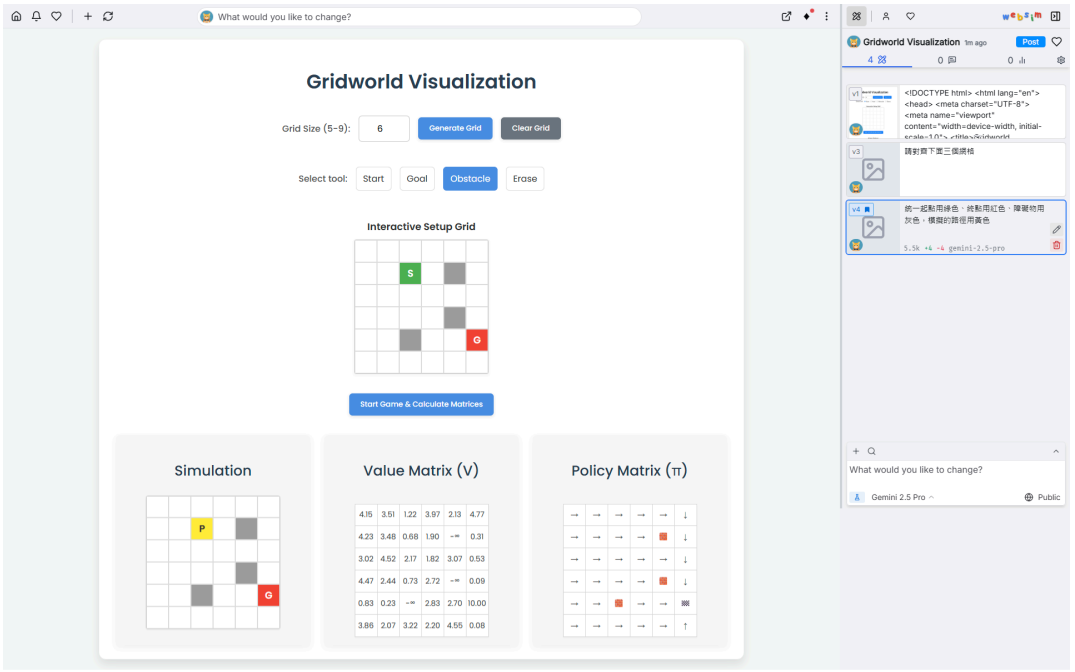
(1) Websim 是什麼？

Websim 是一個線上的網頁工具，我可以將自己撰寫的前端檔案（如 HTML、CSS、JavaScript）上傳到平台上，並透過簡單的說明或操作請求，請它直接幫我修改這些檔案內容。就像使用 RooCode 一樣，Websim 可以根據我輸入的需求，回傳修改後的版本，讓我快速完成前端設計的調整與優化。

在這次作業中，我透過 WebSim 修改了 Gridworld 模擬畫面的樣式與動畫流程，只需簡單描述要改的部分，它就能直接幫我處理程式碼，省下手動修正的時間。

(2) 實際使用 Websim 截圖

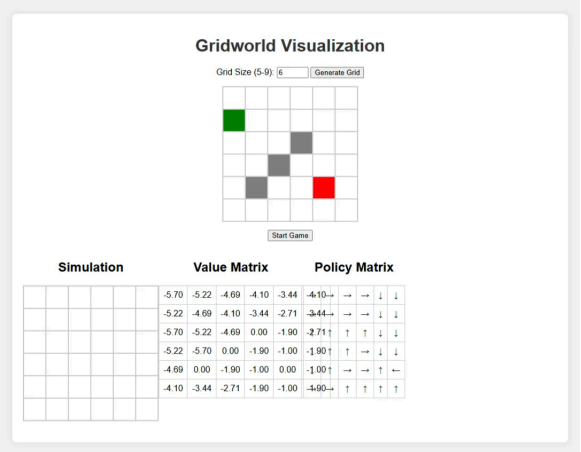
如下方截圖所示，我先上傳自己的前端程式碼，接著在右側對話框輸入需求，WebSim 會根據我的說明直接修改程式碼，並產生新版本的設計，同時，即時預覽更新後的介面，讓我可以快速調整與測試前端效果。



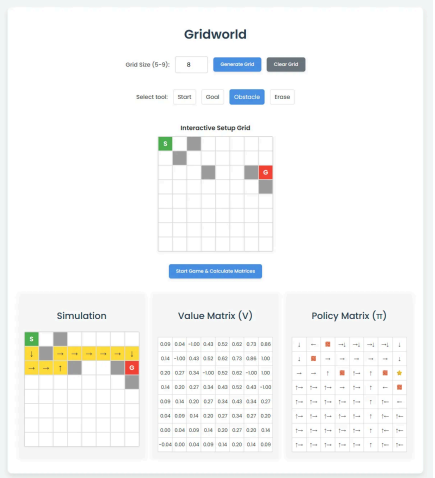
(3) Websim 效果

經由 Websim 協助修改後，整體介面更加直觀美觀，操作流程清楚，並強化了互動設計與數據呈現效果。

- 使用 **Websim 前**



- 使用 **Websim 後**



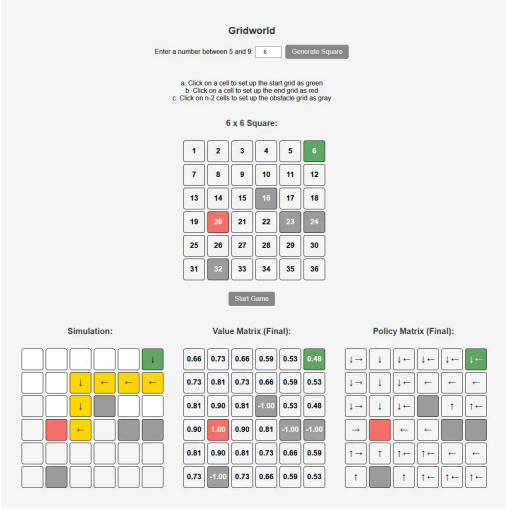
---

4. 結果展示

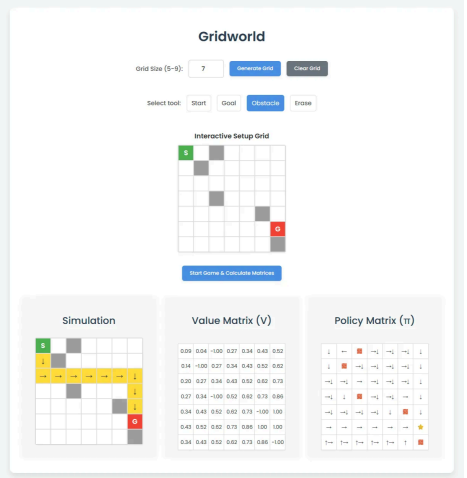
從結果來看，這次用 RooCode 搭配 Websim 重做的效果和原本 HW2 差不多，該有的功能都有做到，不論是模擬、價值矩陣還是策略矩陣都能正確呈現。但在開發效率上提升很多——使用 RooCode 時，我不用再像以前一樣把程式碼複製貼上到 ChatGPT 或其他 AI 工具，只要在 VSCode 裡直接輸入需求，它就會幫我改好本地的程式碼，省下不少時間。

雖然 WebSim 還是要打開瀏覽器，但它能讓我在網頁上直接看到前端畫面的變化，改個 CSS 或排版可以馬上預覽結果，也不用自己慢慢試錯，整體來說開發流程更順、效率也提升很多。這次的重構讓我更有感工具的幫助真的能讓寫程式更輕鬆。

- 原本 HW2 結果



- 使用 RooCode & Websim 結果



---

5. 程式碼說明

▼ (1) `app.py`

這段程式碼是使用 Flask 建立的簡單網站伺服器：

- 當使用者進入首頁 `/` 時，會回傳 `index.html`。
- 靜態資源（如 CSS、JS）則透過 `/static/...` 提供。
- `app.run(debug=True)` 會啟動伺服器並開啟除錯模式。

主要用來展示前端畫面並支援靜態檔案載入。

```
from flask import Flask, render_template, url_for, send_from_directory
import os

app = Flask(__name__, static_folder='static', template_folder='templates')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/static/<path:filename>')
def serve_static(filename):
    return send_from_directory(app.static_folder, filename)

if __name__ == '__main__':
    app.run(debug=True)
```

▼ (2) `index.html`

這是一份用來建立 Gridworld 前端介面的 HTML 網頁，搭配 Flask 使用。簡要說明如下：

- `<head>` 中載入了字體與 CSS 樣式（`style.css`），負責整體排版與視覺設計。
- `<body>` 裡的區塊包含：
  - 標題與控制項（輸入格子大小、生成/清除按鈕）
  - 工具選擇區（設定起點、終點、障礙物、橡皮擦）
  - 互動式格子網格（使用者可點選修改）
  - 模擬與矩陣結果區塊，包含：
    - 模擬結果（Simulation Grid）
    - 價值矩陣（Value Matrix V）
    - 策略矩陣（Policy Matrix π）

最後透過 `<script>` 載入 `script.js`，負責互動邏輯與動畫控制。

整體架構清楚，是用來展示 Gridworld 結果與操作介面的核心 HTML。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Gridworld</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
```

```
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600;700&displ
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <div class="container">
        <h1>Gridworld</h1>

        <div class="controls">
            <label for="gridSize">Grid Size (5-9):</label>
            <input type="number" id="gridSize" min="5" max="9" value="5">
            <button id="generateGrid">Generate Grid</button>
            <button id="clearGrid">Clear Grid</button>
        </div>

        <div class="tool-selection">
            <span>Select tool:</span>
            <label><input type="radio" name="tool" value="start" checked> Start</label>
            <label><input type="radio" name="tool" value="goal"> Goal</label>
            <label><input type="radio" name="tool" value="obstacle"> Obstacle</label>
            <label><input type="radio" name="tool" value="erase"> Erase</label>
        </div>

        <div class="grid-container">
            <p class="grid-label">Interactive Setup Grid</p>
            <div id="grid"></div>
        </div>

        <div class="buttons">
            <button id="startGame">Start Game & Calculate Matrices</button>
        </div>

        <div class="matrices">
            <div class="matrix-container">
                <h2>Simulation</h2>
                <div id="simulationGrid" class="grid-display"></div>
            </div>
            <div class="matrix-container">
                <h2>Value Matrix (V)</h2>
                <table id="valueMatrix"></table>
            </div>
            <div class="matrix-container">
                <h2>Policy Matrix (π)</h2>
                <table id="policyMatrix"></table>
            </div>
        </div>

    </div>

    <script src="{{ url_for('static', filename='script.js') }}"></script>
```

```
</body>
</html>
```

▼ (3) script.js

這是 Gridworld 模擬系統的核心前端腳本，包含互動操作、資料結構、價值迭代演算法與動畫邏輯。以下是各函式的簡要說明：

- `updateObstacleCounter()`：統計目前放置的障礙物數量並更新畫面顯示。
- `updateSelectedTool()`：更新目前選取的工具類型（起點、終點、障礙物、橡皮擦）。
- `initializeGridData(size)`：依照指定尺寸建立空白網格資料。
- `createGridCells(...)`：建立網格畫面，加入格子與互動事件。
- `createTableCells(...)`：建立價值矩陣與策略矩陣表格。
- `updateCellAppearance(...)`：根據格子內容更新外觀與顯示字元。
- `findCellElement(...)`：取得特定格子的 HTML 元素。
- `handleCellClick(...)`：處理點擊格子的互動行為，設定起點、終點、障礙物等。
- `generateNewGrid()`：產生新的互動網格與對應的初始資料。
- `clearInteractiveGrid()`：清除所有使用者設定的起點、終點與障礙物。
- `isValidPosition(r, c)`：檢查座標是否在合法範圍內。
- `valueIteration()`：實作 Value Iteration 算法，計算各格子的最佳價值與策略。
- `findBestPath()`：根據策略矩陣找出從起點到終點的最佳路徑。
- `animatePath(path)`：依據最佳路徑播放動畫，用箭頭顯示移動方向。
- `runGameAndMatrices()`：整合主流程，執行演算法、顯示矩陣並播放動畫。
- `checkPathExists()`：檢查起點與終點之間是否存在有效路徑，避免死路。

整體來說，這份腳本負責將使用者操作轉換為程式邏輯，結合強化學習演算法並透過動畫呈現結果。

```
document.addEventListener('DOMContentLoaded', () => {
    const gridSizeInput = document.getElementById('gridSize');
    const generateGridButton = document.getElementById('generateGrid');
    const clearGridButton = document.getElementById('clearGrid');
    const startGameButton = document.getElementById('startGame');
    const toolRadios = document.querySelectorAll('input[name="tool"]');

    const gridElement = document.getElementById('grid');
    const simulationGridElement = document.getElementById('simulationGrid');
    const valueMatrixTable = document.getElementById('valueMatrix');
    const policyMatrixTable = document.getElementById('policyMatrix');

    const CELL_TYPE = {
        EMPTY: 'empty',
        START: 'start',
        GOAL: 'goal',
        OBSTACLE: 'obstacle',
        PLAYER: 'player'
    };
```

```
// 定義方向動作和對應的箭頭
const ACTIONS = [
    { name: 'UP', r: -1, c: 0, arrow: '↑' },
    { name: 'RIGHT', r: 0, c: 1, arrow: '→' },
    { name: 'DOWN', r: 1, c: 0, arrow: '↓' },
    { name: 'LEFT', r: 0, c: -1, arrow: '←' }
];

const POLICY_ARROWS = {
    UP: '↑',
    RIGHT: '→',
    DOWN: '↓',
    LEFT: '←',
    GOAL: '⭐',
    OBSTACLE: '🧱'
};

let gridSize = 5; // 預設為 5x5
let gridData = [];
let simulationData = [];
let valueData = [];
let policyData = [];

let selectedTool = 'start';
let startPos = null;
let goalPos = null;

function updateObstacleCounter() {
    const count = countObstacles();
    const maxObstacles = gridSize - 2;
    document.getElementById('obstacleCount').textContent = count;
    document.getElementById('maxObstacles').textContent = maxObstacles;
}

function updateSelectedTool() {
    selectedTool = document.querySelector('input[name="tool"]:checked').value;
}

toolRadios.forEach(radio => radio.addEventListener('change', updateSelectedTool));

function initializeGridData(size) {
    const newGridData = [];
    for (let r = 0; r < size; r++) {
        newGridData[r] = [];
        for (let c = 0; c < size; c++) {
            newGridData[r][c] = CELL_TYPE.EMPTY;
        }
    }
    return newGridData;
}
```

```
function createGridCells(element, size, dataArray, isInteractive) {
    element.innerHTML = '';
    element.style.gridTemplateColumns = `repeat(${size}, 1fr)`;
    element.style.gridTemplateRows = `repeat(${size}, 1fr)`;

    for (let r = 0; r < size; r++) {
        for (let c = 0; c < size; c++) {
            const cell = document.createElement('div');
            cell.classList.add('grid-cell');
            cell.dataset.r = r;
            cell.dataset.c = c;
            updateCellAppearance(cell, dataArray[r][c]);
            if (isInteractive) {
                cell.addEventListener('click', handleCellClick);
            }
            element.appendChild(cell);
        }
    }
}

function createTableCells(tableElement, size, dataArray, isPolicy = false) {
    tableElement.innerHTML = '';
    const tbody = tableElement.createTBody();
    for (let r = 0; r < size; r++) {
        const row = tbody.insertRow();
        for (let c = 0; c < size; c++) {
            const cell = row.insertCell();
            if (isPolicy) {
                // 政策矩陣可以包含多個箭頭
                if (Array.isArray(dataArray[r][c])) {
                    cell.textContent = dataArray[r][c].join('');
                } else {
                    cell.textContent = dataArray[r][c] || '';
                }
            } else {
                const val = dataArray[r][c];
                if (val === -Infinity) {
                    cell.textContent = '-∞';
                } else if (val === Infinity) {
                    cell.textContent = '∞';
                } else {
                    // 確保數值格式化為兩位小數
                    cell.textContent = (val !== undefined && val !== null) ? val.toFixed(2) : '0.00';
                }
            }
        }
    }
}

function updateCellAppearance(cellElement, type) {
    cellElement.className = 'grid-cell';
```

```
    cellElement.classList.add(type);
    cellElement.textContent = '';
    if (type === CELL_TYPE.START) cellElement.textContent = 'S';
    else if (type === CELL_TYPE.GOAL) cellElement.textContent = 'G';
    // 移除 Player 的 "P" 標記
    // else if (type === CELL_TYPE.PLAYER) cellElement.textContent = 'P';
}

function findCellElement(gridParent, r, c) {
    return gridParent.querySelector(`.grid-cell[data-r="${r}"][data-c="${c}"]`);
}

function handleCellClick(event) {
    const r = parseInt(event.target.dataset.r);
    const c = parseInt(event.target.dataset.c);

    // 計算當前障礙物數量
    function countObstacles() {
        let count = 0;
        for (let r = 0; r < gridSize; r++) {
            for (let c = 0; c < gridSize; c++) {
                if (gridData[r][c] === CELL_TYPE.OBSTACLE) {
                    count++;
                }
            }
        }
        return count;
    }

    if (selectedTool === CELL_TYPE.START) {
        if (startPos) {
            gridData[startPos.r][startPos.c] = CELL_TYPE.EMPTY;
            updateCellAppearance(findCellElement(gridElement, startPos.r, startPos.c), CELL_TYPE.EM
        }
        startPos = { r, c };
        gridData[r][c] = CELL_TYPE.START;
    } else if (selectedTool === CELL_TYPE.GOAL) {
        if (goalPos) {
            gridData[goalPos.r][goalPos.c] = CELL_TYPE.EMPTY;
            updateCellAppearance(findCellElement(gridElement, goalPos.r, goalPos.c), CELL_TYPE.EMP
        }
        goalPos = { r, c };
        gridData[r][c] = CELL_TYPE.GOAL;
    } else if (selectedTool === CELL_TYPE.ERASE) {
        if (startPos && startPos.r === r && startPos.c === c) startPos = null;
        if (goalPos && goalPos.r === r && goalPos.c === c) goalPos = null;
        gridData[r][c] = CELL_TYPE.EMPTY;
    } else if (selectedTool === CELL_TYPE.OBSTACLE) {
        // 檢查是否已經是障礙物
        if (gridData[r][c] === CELL_TYPE.OBSTACLE) {
            gridData[r][c] = CELL_TYPE.EMPTY;
```

```
        } else {
            // 檢查是否不是起點或終點
            if (gridData[r][c] !== CELL_TYPE.START && gridData[r][c] !== CELL_TYPE.GOAL) {
                // 檢查障礙物數量限制
                const maxObstacles = gridSize - 2;
                if (countObstacles() < maxObstacles) {
                    gridData[r][c] = CELL_TYPE.OBSTACLE;
                } else {
                    alert(`最多只能放置 ${maxObstacles} 個障礙物！`);
                    return; // 不更新單元格
                }
            }
        }
    }

    updateCellAppearance(event.target, gridData[r][c]);
    updateObstacleCounter();
}

function generateNewGrid() {
    gridSize = parseInt(gridSizeInput.value);
    if (gridSize < 5 || gridSize > 9) {
        alert("Grid size must be between 5 and 9.");
        gridSizeInput.value = Math.max(5, Math.min(9, gridSize));
        gridSize = parseInt(gridSizeInput.value);
    }

    gridData = initializeGridData(gridSize);
    startPos = null;
    goalPos = null;
    createGridCells(gridElement, gridSize, gridData, true);

    simulationData = initializeGridData(gridSize);
    createGridCells(simulationGridElement, gridSize, simulationData, false);

    valueData = Array(gridSize).fill().map(() => Array(gridSize).fill(0));
    createTableCells(valueMatrixTable, gridSize, valueData);

    policyData = Array(gridSize).fill().map(() => Array(gridSize).fill(''));
    createTableCells(policyMatrixTable, gridSize, policyData, true);

    updateObstacleCounter(); // 更新計數器
}

function clearInteractiveGrid() {
    gridData = initializeGridData(gridSize);
    startPos = null;
    goalPos = null;
    createGridCells(gridElement, gridSize, gridData, true);
    updateObstacleCounter();
}
```

```
function isValidPosition(r, c) {
    return r >= 0 && r < gridSize && c >= 0 && c < gridSize;
}

function valueIteration() {
    if (!startPos || !goalPos) {
        alert("Please define a Start (S) and a Goal (G) position on the grid.");
        return null;
    }

    // 設定價值迭代參數
    const gamma = 0.9;  // 折扣因子
    const theta = 0.001; // 收斂閾值
    const maxIterations = 1000; // 最大迭代次數

    // 設定獎勵結構
    const goalReward = 1.0;   // 目標獎勵設為 1.0
    const stepCost = -0.04;   // 每步小成本，調整以得到 0-1 範圍內的值

    // 初始化價值矩陣
    let V = Array(gridSize).fill().map(() => Array(gridSize).fill(0));

    // 初始化策略矩陣
    let policy = Array(gridSize).fill().map(() => Array(gridSize).fill().map(() => []));

    // 設定障礙物和目標點
    for (let r = 0; r < gridSize; r++) {
        for (let c = 0; c < gridSize; c++) {
            if (gridData[r][c] === CELL_TYPE.OBSTACLE) {
                V[r][c] = -1.0; // 將障礙物設為 -1.00 而非 -Infinity
                policy[r][c] = POLICY_ARROWS.OBSTACLE;
            } else if (r === goalPos.r && c === goalPos.c) {
                V[r][c] = goalReward; // 目標點設為 1.0
                policy[r][c] = POLICY_ARROWS.GOAL;
            }
        }
    }

    // 價值迭代算法
    let iteration = 0;
    let delta;

    do {
        delta = 0;

        // 對每個狀態進行迭代
        for (let r = 0; r < gridSize; r++) {
            for (let c = 0; c < gridSize; c++) {
                // 跳過障礙物和目標
                if (gridData[r][c] === CELL_TYPE.OBSTACLE ||
```

```
            (r === goalPos.r && c === goalPos.c)) {
            continue;
        }

        const oldValue = V[r][c];
        let maxValue = -Infinity;
        let bestActions = [];

        // 嘗試每個動作
        for (let a = 0; a < ACTIONS.length; a++) {
            const action = ACTIONS[a];
            const newR = r + action.r;
            const newC = c + action.c;

            // 檢查動作是否有效
            if (isValidPosition(newR, newC)) {
                // 如果下一格是障礙物，則視為留在原地
                const nextR = (gridData[newR][newC] === CELL_TYPE.OBSTACLE) ? r : newR;
                const nextC = (gridData[newR][newC] === CELL_TYPE.OBSTACLE) ? c : newC;

                // 獎勵設計
                let reward;
                if (nextR === goalPos.r && nextC === goalPos.c) {
                    reward = goalReward; // 到達目標
                } else if (nextR === r && nextC === c) {
                    reward = -0.1; // 撞到障礙物或牆壁（懲罰更大）
                } else {
                    reward = stepCost; // 一般移動
                }

                const nextValue = reward + gamma * V[nextR][nextC];

                // 更新最大值和最佳動作
                if (Math.abs(nextValue - maxValue) < 1e-6) {
                    bestActions.push(action.arrow);
                } else if (nextValue > maxValue) {
                    maxValue = nextValue;
                    bestActions = [action.arrow];
                }
            }
        }

        // 更新價值和策略
        if (maxValue !== -Infinity) {
            V[r][c] = maxValue;
            // 限制值在範圍內
            if (V[r][c] < -1.0) V[r][c] = -1.0;
            if (V[r][c] > 1.0) V[r][c] = 1.0;

            policy[r][c] = bestActions;
```

```
                // 更新 delta
                delta = Math.max(delta, Math.abs(oldValue - V[r][c]));
              }
            }
          }

          iteration++;

        } while (delta > theta && iteration < maxIterations);

        console.log(`價值迭代在 ${iteration} 次迭代後收斂`);

        // 格式化價值矩陣，使其符合要求的格式
        const formattedV = V.map(row =>
          row.map(val => {
            if (val === -1.0) return -1.0; // 障礙物保持 -1.00
            return Math.round(val * 100) / 100; // 其他值四捨五入到兩位小數
          })
        );

        return { valueMatrix: formattedV, policyMatrix: policy };
      }

      function findBestPath() {
        if (!startPos || !goalPos) return [];

        // 執行價值迭代
        const result = valueIteration();
        if (!result) return [];

        const { policyMatrix } = result;

        // 找出從起點到終點的最佳路徑
        let path = [];
        let current = { ...startPos };
        let maxSteps = gridSize * gridSize; // 防止無限循環
        let steps = 0;

        // 將起點加入路徑
        path.push({ ...current });

        // 循環直到達到終點或達到最大步數
        while (!(current.r === goalPos.r && current.c === goalPos.c) && steps < maxSteps) {
          const currentPolicy = policyMatrix[current.r][current.c];

          // 如果策略為空或不是陣列，則停止
          if (!currentPolicy || !Array.isArray(currentPolicy) || currentPolicy.length === 0) {
            break;
          }

          // 選擇第一個可用的動作
```

```
            const nextMoveArrow = currentPolicy[0];
            let nextMove = null;

            // 找到箭頭對應的動作
            for (const action of ACTIONS) {
                if (action.arrow === nextMoveArrow) {
                    nextMove = action;
                    break;
                }
            }

            // 如果沒有有效動作，則停止
            if (!nextMove) break;

            // 計算下一步的位置
            const newR = current.r + nextMove.r;
            const newC = current.c + nextMove.c;

            // 檢查下一步是否有效
            if (!isValidPosition(newR, newC) ||
                gridData[newR][newC] === CELL_TYPE.OBSTACLE) {
                break;
            }

            // 更新當前位置並加入路徑
            current = { r: newR, c: newC };
            path.push({ ...current });
            steps++;
        }

        return path;
    }

    function animatePath(path) {
        // 深拷貝原始網格數據
        simulationData = JSON.parse(JSON.stringify(gridData));
        createGridCells(simulationGridElement, gridSize, simulationData, false);

        // 逐步顯示路徑
        path.forEach((point, index) => {
            setTimeout(() => {
                if ((point.r === startPos.r && point.c === startPos.c) ||
                    (point.r === goalPos.r && point.c === goalPos.c)) {
                    return; // 跳過起點和終點
                }

                const cell = findCellElement(simulationGridElement, point.r, point.c);
                cell.className = 'grid-cell player';

                // 如果不是最後一個點，添加箭頭指示方向
                if (index < path.length - 1) {
```

```
            const nextPoint = path[index + 1];
            let direction = '';

            // 計算方向
            if (nextPoint.r < point.r) direction = POLICY_ARROWS.UP;
            else if (nextPoint.r > point.r) direction = POLICY_ARROWS.DOWN;
            else if (nextPoint.c < point.c) direction = POLICY_ARROWS.LEFT;
            else if (nextPoint.c > point.c) direction = POLICY_ARROWS.RIGHT;

            // 設置箭頭
            cell.textContent = direction;
        }
    }, index * 300); // 每 300ms 顯示一步
  });
}

function runGameAndMatrices() {
  if (!startPos || !goalPos) {
    alert("Please define a Start (S) and a Goal (G) position on the grid.");
    return;
  }

  // 添加路徑檢查
  if (!checkPathExists()) {
    alert("No valid path exists from start to goal. Please adjust the obstacles.");
    return;
  }

  // 執行價值迭代
  const result = valueIteration();
  if (!result) return;

  const { valueMatrix, policyMatrix } = result;

  // 顯示價值矩陣和策略矩陣
  valueData = valueMatrix;
  policyData = policyMatrix;

  createTableCells(valueMatrixTable, gridSize, valueData);
  createTableCells(policyMatrixTable, gridSize, policyData, true);

  // 找出最佳路徑並顯示
  const bestPath = findBestPath();

  // 動畫顯示路徑
  animatePath(bestPath);
}
function checkPathExists() {
  if (!startPos || !goalPos) return false;

  // 使用廣度優先搜索檢查路徑
```

```
    const queue = [{ ...startPos }];
    const visited = {};
    visited[`${startPos.r},${startPos.c}`] = true;

    while (queue.length > 0) {
      const current = queue.shift();

      // 如果到達目標，返回 true
      if (current.r === goalPos.r && current.c === goalPos.c) {
        return true;
      }

      // 嘗試四個方向
      for (const action of ACTIONS) {
        const newR = current.r + action.r;
        const newC = current.c + action.c;

        // 檢查是否有效且未訪問
        if (isValidPosition(newR, newC) &&
          gridData[newR][newC] !== CELL_TYPE.OBSTACLE &&
          !visited[`${newR},${newC}`]) {
          queue.push({ r: newR, c: newC });
          visited[`${newR},${newC}`] = true;
        }
      }
    }

    return false; // 沒有找到路徑
  }

  // 添加事件監聽器
  generateGridButton.addEventListener('click', generateNewGrid);
  clearGridButton.addEventListener('click', clearInteractiveGrid);
  startGameButton.addEventListener('click', runGameAndMatrices);

  // 初始化
  updateSelectedTool();
  generateNewGrid();
});
```

▼ (4) `style.css`

這份 CSS 負責整體畫面風格與互動式排版，強調可讀性與現代設計感。以下為主要設計要點：

- **整體排版**：採用 `Poppins` 字體，搭配淺灰底與卡片式白色容器，整體感清爽。
- **互動網格與格子樣式**：以 `.grid-cell` 為主體，支援 hover 效果、動畫標記與類別區分（起點、終點、障礙物、動畫玩家等）。
- **控制面板與工具選擇區**：以 `flexbox` 進行整齊排列，並透過選中樣式強化可視性（藍色高亮背景、白字）。
- **矩陣表格樣式**：統一格子大小與對齊，強調邊框與配色，清楚呈現 Value/Policy 矩陣。
- **按鈕與輸入框**：所有按鈕皆有 hover 動畫與主題色調（深藍、灰），輸入框則有聚焦高亮效果。

整體設計兼顧實用與視覺美感，讓使用者能輕鬆操作並清楚理解模擬結果。
定義格子樣式、箭頭方向顏色、黃金路徑樣式與整體排版，讓模擬畫面更清晰。

```
body {
    font-family: 'Poppins', Arial, sans-serif;
    margin: 0;
    padding: 20px;
    background-color: #f4f7f6; /* Lighter, slightly greenish gray */
    color: #333;
    display: flex;
    flex-direction: column;
    align-items: center;
    min-height: 100vh;
    box-sizing: border-box;
}

.container {
    background-color: #ffffff;
    padding: 25px 30px;
    border-radius: 12px;
    box-shadow: 0 8px 16px rgba(0,0,0,0.1);
    width: 90%;
    max-width: 1100px; /* Slightly increased max-width */
}

h1, h2 {
    text-align: center;
    color: #2c3e50; /* Darker, more modern blue-gray */
    margin-bottom: 25px;
}

h1 {
    font-size: 2.2em;
    font-weight: 600;
}

h2 {
    font-size: 1.6em;
    font-weight: 500;
    margin-top: 30px; /* Added margin top for separation */
}

.controls, .tool-selection, .buttons {
    margin-bottom: 25px;
    display: flex;
    gap: 15px; /* Increased gap */
    align-items: center;
    flex-wrap: wrap;
    justify-content: center;
    padding: 10px 0; /* Added some vertical padding */
}
```

```css
.controls label, .tool-selection span {
    font-weight: 500;
    color: #555;
}

.tool-selection label {
    cursor: pointer;
    padding: 8px 12px;
    border: 1px solid #ddd;
    border-radius: 6px;
    transition: background-color 0.3s, color 0.3s, border-color 0.3s;
    display: flex;
    align-items: center;
    gap: 5px;
}

.tool-selection input[type="radio"] {
    margin-right: 5px;
    accent-color: #4A90E2; /* Match primary color */
}

.tool-selection input[type="radio"] {
    display: none; /* Hide the actual radio button */
}

.tool-selection label:has(input[type="radio"]:checked) {
    background-color: #4A90E2;
    color: white;
    border-color: #4A90E2;
}

.grid-container {
    margin-bottom: 25px;
    display: flex;
    flex-direction: column;
    align-items: center;
}

.grid-label {
    font-weight: 600; /* Bolder label */
    margin-bottom: 10px; /* Increased margin */
    font-size: 1.1em;
    color: #333;
}

#grid, .grid-display {
    display: grid;
    border: 1px solid #d0d0d0; /* Slightly darker border for definition */
    background-color: #fdfdfd;
    border-radius: 4px; /* Subtle rounding */
```

```
    box-shadow: 0 2px 4px rgba(0,0,0,0.05); /* Subtle shadow */
}

.grid-cell {
    width: 40px;
    height: 40px;
    border: 1px solid #e0e0e0; /* Lighter cell borders */
    display: flex;
    justify-content: center;
    align-items: center;
    font-size: 1.1em; /* Slightly larger icons/text */
    font-weight: 600;
    cursor: pointer;
    box-sizing: border-box;
    transition: background-color 0.2s ease-in-out, transform 0.1s ease;
}

#grid .grid-cell:hover {
    background-color: #f0f0f0; /* Subtle hover for interactive grid */
    transform: scale(1.05);
}

.grid-cell.empty { background-color: #fff; }
.grid-cell.start { background-color: #4CAF50; color: white; content: 'S'; } /* Green */
.grid-cell.goal { background-color: #F44336; color: white; content: 'G'; }  /* Red */
.grid-cell.obstacle { background-color: #9E9E9E; color: #fff } /* Gray */
.grid-cell.player {
    background-color: #ffdb3b;
    animation: pulse 1.5s infinite alternate;
}

.matrices-and-simulation {
    display: flex;
    justify-content: space-between;
    align-items: flex-start; /* Align items to the top */
    width: 100%;
    margin-bottom: 25px;
}

.simulation-container {
    width: 30%;
}

.matrices {
    display: flex;
    flex-direction: row; /* 改為水平排列 */
    justify-content: space-between; /* 平均分配空間 */
    gap: 20px; /* 元素之間的間距 */
    margin-top: 30px;
    width: 100%; /* 確保使用全寬 */
    flex-wrap: wrap; /* 在小螢幕上允許換行 */
```

```css
}

.matrix-container {
    flex: 1; /* 平均分配空間 */
    min-width: 280px; /* 最小寬度 */
    max-width: calc(33.333% - 20px); /* 確保在大螢幕上每個不超過 1/3 寬度 */
    display: flex;
    flex-direction: column;
    align-items: center;
    background-color: #f9f9f9;
    padding: 15px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.05);
    margin-bottom: 20px;
}

.matrix-container {
    flex: 1;
    min-width: 280px; /* Slightly increased min-width */
    display: flex;
    flex-direction: column;
    align-items: center;
    background-color: #f9f9f9; /* Light background for matrix containers */
    padding: 15px;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.05);
}

.obstacle-counter {
    margin: 10px 0;
    font-weight: 500;
    color: #555;
    text-align: center;
}

table {
    border-collapse: collapse;
    margin-top: 15px; /* Increased margin */
    background-color: #fff;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
}

th, td {
    border: 1px solid #dadada; /* Lighter border for table cells */
    width: 40px; /* Changed from 42px to match .grid-cell */
    height: 40px; /* Changed from 42px to match .grid-cell */
    text-align: center;
    vertical-align: middle;
    font-size: 0.85em; /* Adjusted font size */
    box-sizing: border-box;
}
```

```css
th {
    background-color: #e9ecef; /* Light gray for table headers */
    font-weight: 600;
    color: #333;
}

#simulationGrid .grid-cell {
    cursor: default; /* Simulation grid is not interactive by click */
}

#simulationGrid .grid-cell:hover {
    transform: none; /* No hover transform for simulation grid */
}

button {
    padding: 10px 20px; /* Increased padding */
    font-family: 'Poppins', sans-serif; /* Consistent font */
    font-weight: 500; /* Medium weight */
    background-color: #4A90E2; /* Primary blue */
    color: white;
    border: none;
    border-radius: 6px; /* Rounded corners */
    cursor: pointer;
    transition: background-color 0.3s, box-shadow 0.3s, transform 0.2s;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

button:hover {
    background-color: #357ABD; /* Darker blue on hover */
    box-shadow: 0 4px 8px rgba(0,0,0,0.15);
    transform: translateY(-1px);
}

button:active {
    background-color: #2a6496;
    transform: translateY(0px);
    box-shadow: 0 1px 2px rgba(0,0,0,0.1);
}

#clearGrid {
    background-color: #6c757d; /* Grayish color */
}
#clearGrid:hover {
    background-color: #5a6268;
}
#clearGrid:active {
    background-color: #545b62;
}

input[type="number"] {
```

```
    padding: 10px; /* Increased padding */
    border: 1px solid #ccc;
    border-radius: 6px;
    width: 70px; /* Increased width */
    font-family: 'Poppins', sans-serif;
    font-size: 1em;
    text-align: center;
    transition: border-color 0.3s, box-shadow 0.3s;
}

input[type="number"]:focus {
    border-color: #4A90E2;
    box-shadow: 0 0 0 0.2rem rgba(74, 144, 226, 0.25);
    outline: none;
}
```