Kakuro Game Documentation

David Sequera

October 2023

Contents

1 Introduction

This document provides a description of the Kakuro game, including its rules, the technologies used for implementation, a detailed interface description, and how the game board interacts with React.

2 Rules of the Game

Kakuro is a number puzzle game played on a grid. The objective is to fill in the grid with numbers from 1 to 9 such that:

- Each row and column must contain each number from 1 to 9 exactly once.
- The numbers in each **shaded** (stack) cell must sum up to the value given in that cell.
- No two cells in the same row or column can contain the same number.

3 Technologies Used

The Kakuro game is implemented using the following technologies:

- **Framework:** Next.js
- **Language:** TypeScript
- **Reactive Programming:** RxJS

4 Interface Description

The game interface is built around a `GameBoard` class that represents the game board. The board consists of a 2-dimensional table with three types of cells:

1. **Blocked Cell:** These cells are not part of the game and are usually indicated with a different color.
2. **Input Cell:** These cells are user-editable and can contain values from 1 to 9.
3. **Stack Cell:** These cells represent the sums of rows and columns. They hold a tuple of two elements, indicating the sum for the row and column.

4.1 Board Initialization

The board is initialized as follows:

1. All cells are filled with random values from 1 to 9.
2. Stack cells are selected, and their row and column sums are calculated and stored in the tuple.
3. Input cell values are cleared, making them editable for the player.

Listing 1: Board Creation Algorithm

```
export class GameBoard {
    private _board: Array<Array<any>> = [];
    private _solution: Array<Array<number>> = [];
    private _types: Array<Array<TypeCell>> = [];
    private _subjects: Array<Array<Subject<cell>
                    | [Subject<cell>, Subject<cell> ]>> = [];

    private _initBoard(r: number, c: number): void {
        // Set the input cells
        for (let i = 0; i < r; i++) {
            this._board[i] = [];
            this._types[i] = [];
            this._subjects[i] = [];
            for (let j = 0; j < c; j++) {
                this._board[i][j] = Math.round(Math.random() * 8 + 1);
                this._types[i][j] = TypeCell.INPUT;
            }
        }
        // Set the stack cells
        for (let i = 0; i < r; i++) {
            this._board[i][0] = [0, 0];
            this._types[i][0] = TypeCell.STACK;
            for (let j = 1; j < c; j++) {
                this._board[i][0][0] += this._board[i][j] as number;
            }
        }
        for (let j = 0; j < c; j++) {
            this._board[0][j] = [0, 0];
            this._types[0][j] = TypeCell.STACK;
            for (let i = 1; i < r; i++) {
                this._board[0][j][1] += this._board[i][j] as number;
            }
        }
        // Set the blocked cells
        this._types[0][0] = TypeCell.BLOCKED;
        // Save one copy of the solution
```

```
        this._solution = this._board.slice(0);
        // clean the board
        for (let i = 1; i < r; i++) {
            for (let j = 1; j < c; j++) {
                this._board[i][j] = 0;
            }
        }
        // Set the subjects
        for (let i = 1; i < r; i++) {
            for (let j = 1; j < c; j++) {
                if (this._types[i][j] === TypeCell.INPUT)
                    this._subjects[i][j] = new Subject<cell>();
            }
        }
        // Set the subjects of the black cells
        for (let i = 0; i < r; i++) {
            for (let j = 0; j < c; j++) {
                if (this._types[i][j] === TypeCell.STACK)
                    this._subjects[i][j] = new Subject<cell>();
            }
        }
        // subscribe to the black cells
        for (let i = 0; i < r; i++) {
            for (let j = 0; j < c; j++) {
                this.subscribeBlackCells(this.getCell(i, j));
            }
        }
    }
}
```

5 Board Communication with React

The board uses the Observer pattern with Subjects to communicate with React. Each cell has a subject associated with it:

- **Input Cells:** Communicate changes in their values.
- **Stack Cells:** Subscribe to each input cell and communicate changes in the sums of their corresponding rows and columns.

Listing 2: Input Cell Comunication

```
// Method of GameBoard Class
setCell(i: number, j: number, value: cellValue): void {
    this._board[i][j] = value;

    const subject = this._subjects[i][j];
    if (Array.isArray(subject)) {
        subject.forEach(s => s.next(this.getCell(i, j)));
    } else {
        subject.next(this.getCell(i, j));
    }
}
```

Listing 3: Input Cell Comunication

```
// Methods of GameBoard Class
private subscribeBlackCells(cell: cell): void {
    if(cell.type !== TypeCell.STACK) return;
```

```
        const input_cells = this.getStackCells(cell)
        const rowCells = input_cells[0];
        const colCells = input_cells[1];
        // subscribe to the rows and columns
        const observable_rows  = merge(...rowCells.map(c => c.subject));
        const observable_cols = merge(...colCells.map(c => c.subject));
        // subscribe to the rows and columns
        const subjectRow = new Subject<cell>();
        const subjectCol = new Subject<cell>();

        observable_rows.subscribe(subjectRow);
        observable_cols.subscribe(subjectCol);

        // save the subjects
        this._subjects[cell.i][cell.j] = [subjectRow, subjectCol];
    }
    // when the cell is black, and the sum of the row and the sum of the column are equ
    winStackCell(cell: cell): boolean {
        const [rowValue, colValue] = cell.value as [number, number];
        const [rowCells, colCells] = this.getStackCells(cell);
        const rowSum = rowCells.reduce((a, b) => a + (b.value as number), 0);
        const colSum = colCells.reduce((a, b) => a + (b.value as number), 0);
        console.log('[winStackCell]',rowSum, colSum);
        return rowSum === rowValue && colSum === colValue;
    }

    private getStackCells(cell: cell): [Array<cell>, Array<cell>] {
        const [i, j] = [cell.i, cell.j];
        const rowCells: Array<cell> = [];
        const colCells: Array<cell> = [];
        // get rows
        for (let k = 1; k < this.c; k++) {
            if (this._types[i][k] === TypeCell.INPUT) {
                rowCells.push(this.getCell(i, k));
            }
        }
        // get columns
        for (let k = 1; k < this.r; k++) {
            if (this._types[k][j] === TypeCell.INPUT) {
                colCells.push(this.getCell(k, j));
                // console.log('[getStackCells columns]',k, j, cell);
            }
        }
        console.log(`[getStackCells] ${cell.i} ${cell.j}`,rowCells, colCells);
        return [rowCells, colCells];
    }
```

Whenever a cell's value changes, it notifies its subject, and React updates the UI accordingly. The board also performs checks to ensure that the sums of input cells in rows and columns match the values in stack cells to validate the game's progress.

6 Game Modes

The game is meant to be played with modes like 3x3, 5x5, and 10x10

7 Conclusion

This document provides an overview of the Kakuro game, including its rules, the technologies used for implementation, and an in-depth description of the game's interface and communication with React.