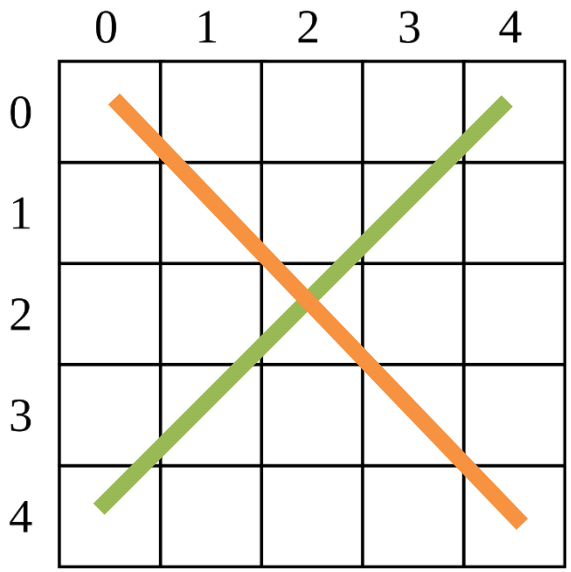
Coding Board Games

coding academy

Matrix as a game board

When coding board games, it is often useful to use a matrix



Chess board

Here is a chess board represented as a matrix of strings.

```
var board = [
    ['♜', '♞', '♝', '♛', '♚', '♝', '♞', '♜'],
    ['♟', '♟', '♟', '♟', '♟', '♟', '♟', '♟'],
    ['' , '' , '' , '' , '' , '' , '' , ''],
    ['' , '' , '' , '' , '' , '' , '' , ''],
    ['' , '' , '' , '' , '' , '' , '' , ''],
    ['' , '' , '' , '' , '' , '' , '' , ''],
    ['♙', '♙', '♙', '♙', '♙', '♙', '♙', '♙'],
    ['♖', '♘', '♗', '♕', '♔', '♗', '♘', '♖']
]
console.log('board', board)
```

```
board ▼ (8) [Array(8), Array(8), Array(8), Array(8), Array(8), Array(8), Array(8), Array(8)]
    ▶ 0: (8) ['♜', '♞', '♝', '♛', '♚', '♝', '♞', '♜']
    ▶ 1: (8) ['♟', '♟', '♟', '♟', '♟', '♟', '♟', '♟']
    ▶ 2: (8) ['', '', '', '', '', '', '', '']
    ▶ 3: (8) ['', '', '', '', '', '', '', '']
    ▶ 4: (8) ['', '', '', '', '♙', '', '', '']
    ▶ 5: (8) ['', '', '', '', '', '', '', '']
    ▶ 6: (8) ['♙', '♙', '♙', '♙', '', '♙', '♙', '♙']
    ▶ 7: (8) ['♖', '♘', '♗', '♕', '♔', '♗', '♘', '♖']
      length: 8
```

Chess board

When the matrix holds simple values such as strings, we can also use `console.table` :

```
var board = [
    ['♜', '♞', '♝', '♛', '♚', '♝', '♞', '♜'],
    ['♟', '♟', '♟', '♟', '♟', '♟', '♟', '♟'],
    ['', '', '', '', '', '', '', ''],
    ['', '', '', '', '', '', '', ''],
    ['', '', '', '', '', '', '', ''],
    ['', '', '', '', '', '', '', ''],
    ['♙', '♙', '♙', '♙', '♙', '♙', '♙', '♙'],
    ['♖', '♘', '♗', '♕', '♔', '♗', '♘', '♖']
]
console.table(board)
```

(ind.	0	1	2	3	4	5	6	7
0	'♜'	'♞'	'♝'	'♛'	'♚'	'♝'	'♞'	'♜'
1	'♟'	'♟'	'♟'	'♟'	'♟'	'♟'	'♟'	'♟'
2	''	''	''	''	''	''	''	''
3	''	''	''	''	''	''	''	''
4	''	''	''	''	''	''	''	''
5	''	''	''	''	''	''	''	''
6	'♙'	'♙'	'♙'	'♙'	'♙'	'♙'	'♙'	'♙'
7	'♖'	'♘'	'♗'	'♕'	'♔'	'♗'	'♘'	'♖'

Moving a piece on the Chess board

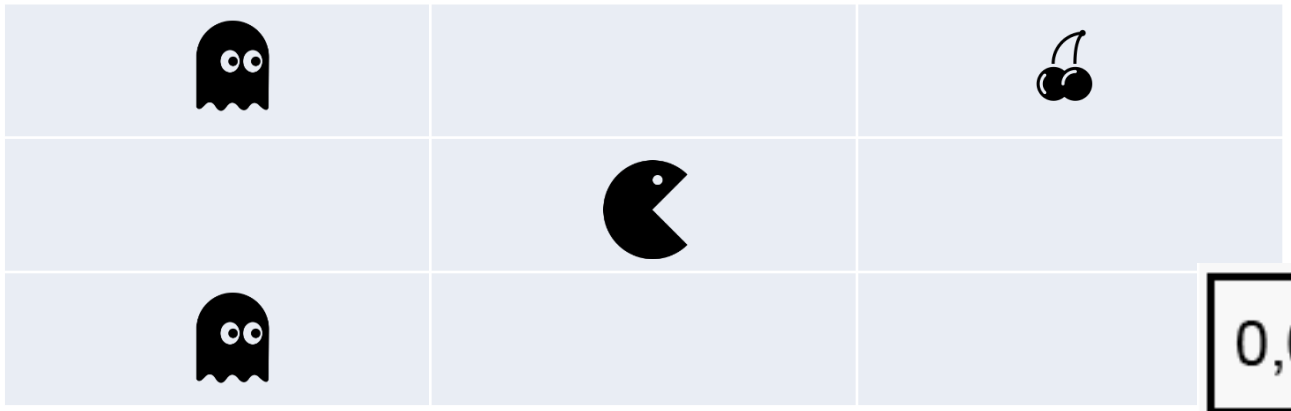

```
// White King's Pawn forward 2
board[4][4] = board[6][4]
board[6][4] = ''

// Black Left Rook's Pawn forward 1
board[2][0] = board[1][0]
board[1][0] = ''
```

coding academy

Matrix Neighbors

When dealing with matrixes, it is sometimes needed to look around a certain cell

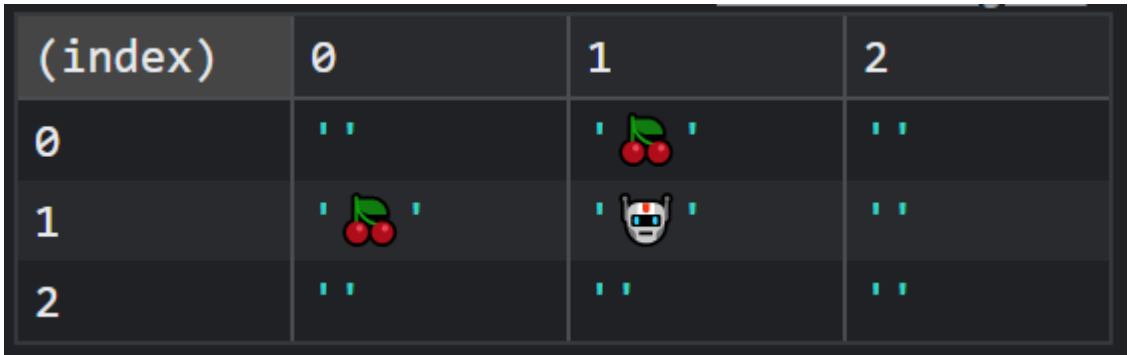
- Generally, cells have 8 neighbors
- Cells at the edges have less..



0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Let's create such a Matrix

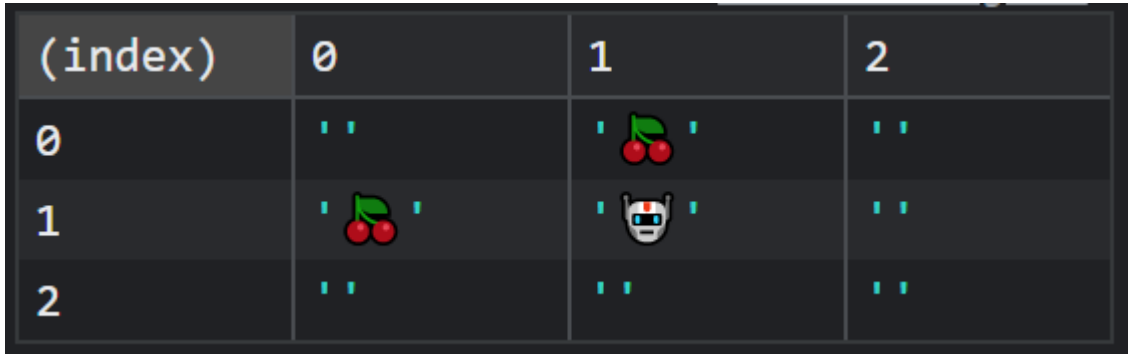
```
const FOOD = ''
const EMPTY = ''
const HERO = ''

var gBoard = createBoard()
console.table(gBoard)

function createBoard() {
    var board = []
    for (var i = 0; i < 3; i++) {
        board[i] = []
        for (var j = 0; j < 3; j++) {
            board[i][j] = (Math.random() > 0.7) ? FOOD : EMPTY
        }
    }
    board[1][1] = HERO
    return board
}
```



(index)	0	1	2
0	''	''	''
1	''	''	''
2	''	''	''

Let's check for food (1/3)

```
var count = countFoodAround(gBoard, 1, 1)
console.log('Found', count, ' food around')

function countFoodAround(board, rowIdx, colIdx) {
    var foodCount = 0
    for (var i = rowIdx-1; i <= rowIdx+1; i++) {
        for (var j = colIdx-1; j <= colIdx+1; j++) {
            var currCell = board[i][j]
            if (currCell === FOOD) foodCount++
        }
    }
    return foodCount
}
```

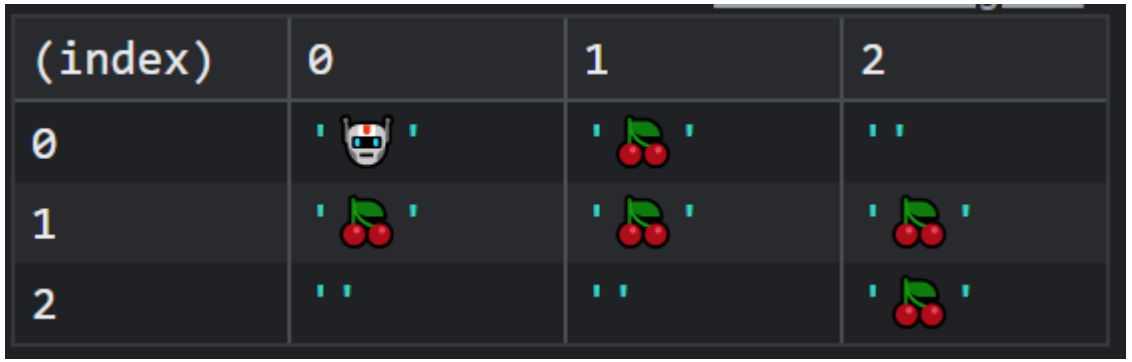


(index)	0	1	2
0	''	'🍒'	''
1	'🍒'	'🤖'	''
2	''	''	''

Mind the edge

- While looking around a cell, we need to be careful at the edges
- For example, trying to count food for cell 0, 0 we will get an error such as:

```
var count = countFoodAround(gBoard, 0, 0)
```



(index)	0	1	2
0	''	''	''
1	''	''	''
2	''	''	''

```
Uncaught TypeError: Cannot read properties of undefined (reading '-1')
    at countFoodAround (15-matrix.js:29:36)
    at 15-matrix.js:21:13
```

Let's improve our loop (2/3)

```
var count = countFoodAround(gBoard, 0, 0)
console.log('Found', count, ' food around')

function countFoodAround(board, rowIdx, colIdx) {
    var foodCount = 0
    for (var i = rowIdx - 1; i <= rowIdx + 1; i++) {
        if (i < 0 || i >= board.length) continue
        for (var j = colIdx - 1; j <= colIdx + 1; j++) {

            if (j < 0 || j >= board[0].length) continue
            var currCell = board[i][j]
            if (currCell === FOOD) foodCount++
        }
    }
    return foodCount
}
```

(index)	0	1	2
0	'🤖'	'🍒'	''
1	'🍒'	'🍒'	'🍒'
2	''	''	'🍒'

The Neighbors Loop (3/3)

Usually, we will skip the middle cell (its not a neighbor) within the Neighbors loop – it looks like that:

```
var count = countFoodAround(gBoard, 0, 0)
console.log('Found', count, ' food around me')

function countFoodAround(board, rowIdx, colIdx) {
    var foodCount = 0
    for (var i = rowIdx - 1; i <= rowIdx + 1; i++) {
        if (i < 0 || i >= board.length) continue
        for (var j = colIdx - 1; j <= colIdx + 1; j++) {
            if (i === rowIdx && j === colIdx) continue
            if (j < 0 || j >= board[0].length) continue
            var currCell = board[i][j]
            if (currCell === FOOD) foodCount++
        }
    }
    return foodCount
}
```

coding academy

Matrix Neighbors

Lets find the **best position** on the board

```
var bestPos = findBestPos(gBoard)
console.log('Best pos:', bestPos)
```

```
Best pos: ▶ {i: 3, j: 3}
```

```
function findBestPos(board) {
    var maxFoodCount = 0
    var bestPos = null
    for (var i = 0; i < board.length; i++) {
        for (var j = 0; j < board[0].length; j++) {
            if (board[i][j] === FOOD) continue
            var count = countFoodAround(board, i, j)
            if (count > maxFoodCount) {
                maxFoodCount = count
                bestPos = { i: i, j: j }
            }
        }
    }
    return bestPos
}
```

(index)	0	1	2	3
0	''	''	''	''
1	''	''	''	''
2	''	''	'🍒'	'🍒'
3	''	''	'🍒'	''

Coding Board Games

Matrix as a <table>

We may use an HTML Table to present a game board that is defined in a matrix.

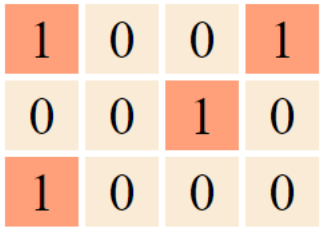
```
<tr>
    <td class="cell taken" data-i="0" data-j="0" onclick="onCellClick(0, 0)">
        1
    </td>

    <td class="cell " data-i="0" data-j="1" onclick="onCellClick(0, 1)">
        0
    </td>
```

```
<style>
.cell {
    width: 20px;
    height: 20px;
    background-color: antiquewhite;
    text-align: center;
}
.taken {
    background-color: lightsalmon;
}
</style>
```

1	0	0	1
0	0	1	0
1	0	0	0

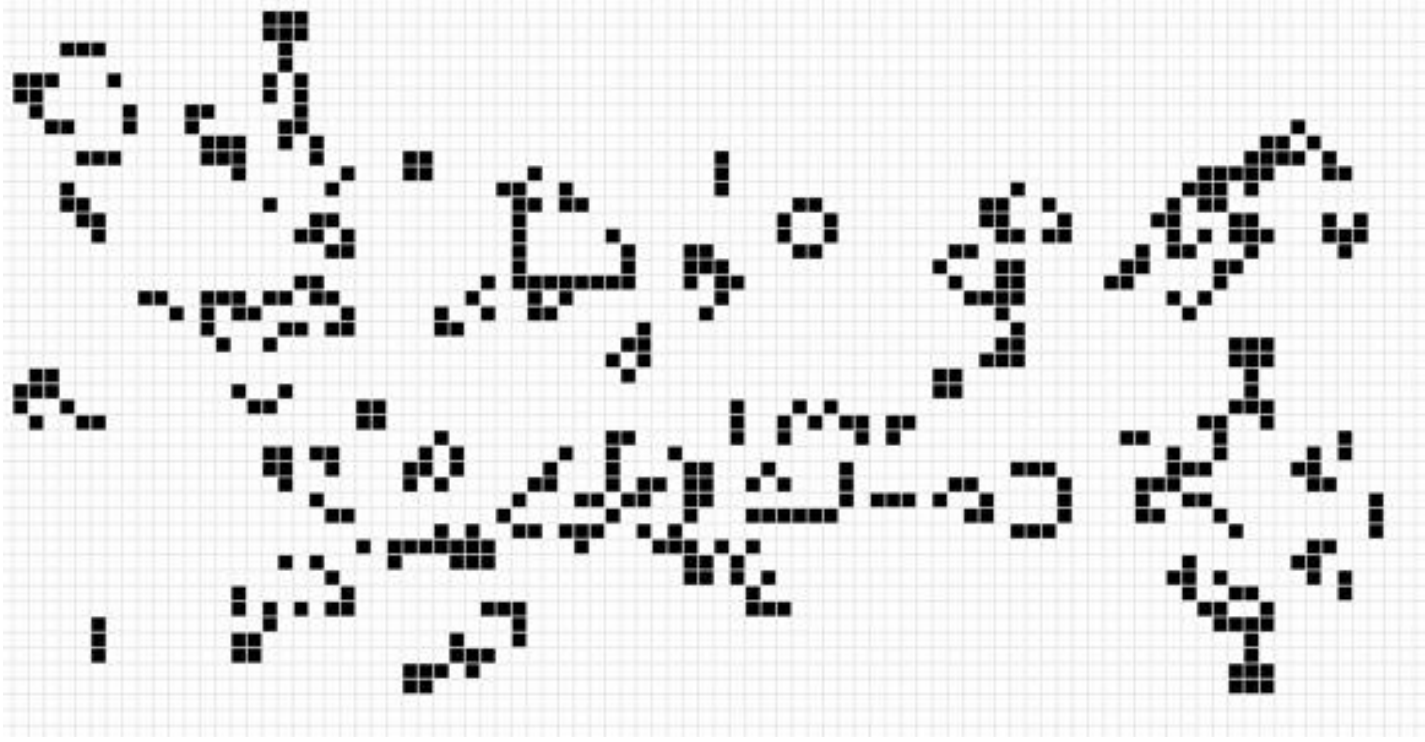
coding academy

```
<body onload="onInit()">
    <table>
        <tbody class="board"></tbody>
    </table>
    <script>
        var gBoard = [
            [1, 0, 0, 1],
            [0, 0, 1, 0],
            [1, 0, 0, 0],
        ]
        function onInit() {
            renderBoard(gBoard)
        }
        function renderBoard(board) {
            var strHTML = ''
            for (var i = 0; i < board.length; i++) {
                strHTML += '<tr>'
                for (var j = 0; j < board[0].length; j++) {
                    var currCell = board[i][j]
                    var cellClass = (board[i][j]) ? 'taken' : ''
                    var cellData = 'data-i="' + i + '" data-j="' + j + '"'
                    strHTML += `
                        <td class="cell ${cellClass}" ${cellData} onclick="onCellClick(${i}, ${j})">
                            ${currCell}
                        </td>
                    `

                }
                strHTML += '</tr>'
            }
            var elBoard = document.querySelector('.board')
            elBoard.innerHTML = strHTML
        }
    </script>
</body>
```



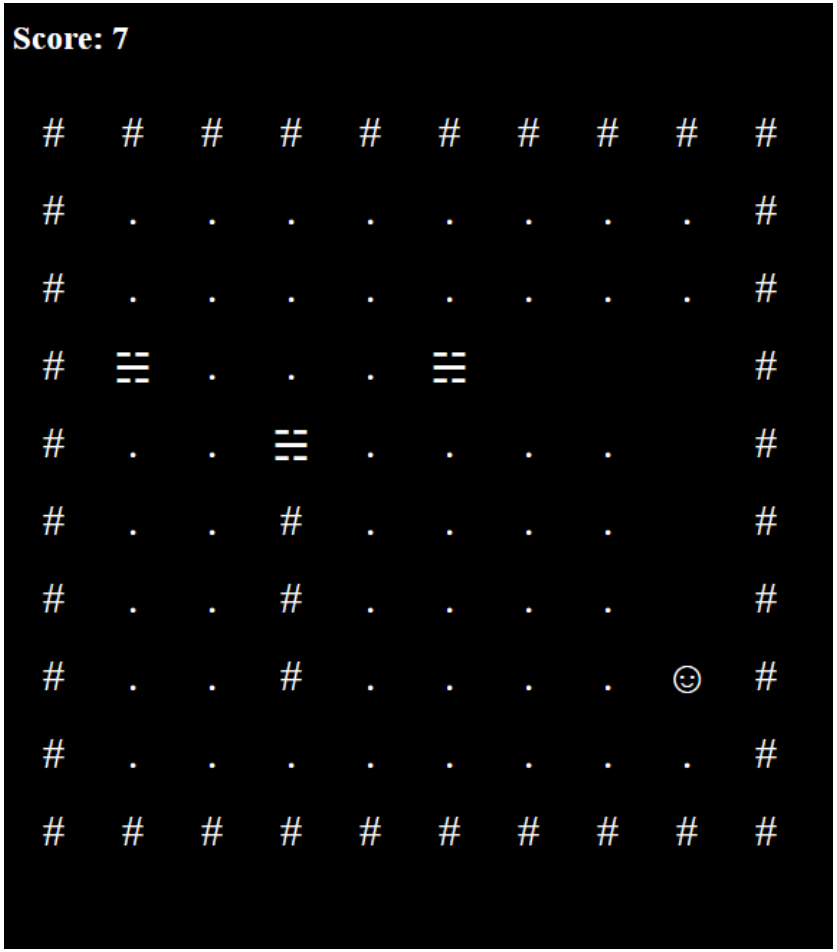
Game of life

Lets display Game-of-life in an HTML Table



Lets Plan Pacman

- Identify global data structures:
  - gBoard
  - gPacman
  - gGhosts
  - gGame




```
const WALL = '#'
const FOOD = '.'
const EMPTY = ' '
```

```
pacman = {
    location: {
        i: 3,
        j: 5
    },
    isSuper: false
}
```

```
ghost = {
    location: {
        i: 3,
        j: 3
    },
    currCellContent: FOOD
}
```

coding academy

Coding Board Games