Aufgabe 1: Arukone

Team-ID: 00265

Teilnahme-ID: 69306

Team-Name: Die Debugger

Bearbeiter dieser Aufgabe:
Abulfasl Ahmadi

19. November 2023

Inhaltsverzeichnis

1 Lösungsidee und Umsetzung

Vorweg sei gesagt, dass die Umsetzung in Java Script ES6 erfolgte. Die Lösungsidee ist zusätzich in Pseudocode beschrieben. Der Quellcode ist im Anhang zu finden.

Für die Eingabe n gilt: $n \in \mathbb{N}; n \geq 4$

Es wird zunächst ein zwei Dimensionales n x n großes Gitter mit n^2 vielen Feldern erstellt.

Des weiteren werden für die möglichen Zahlenpaare x ($x \in \mathbb{N}; n > x \geq \frac{n}{2}$) zufällige y- und x-Koordinaten für das Gitter zwischen 0 und $n-1$ generiert. Wenn die generieten Koordinaten im Gitter für $x_1 \vee x_2$ bereits vergeben sind, wird für das Zahlenpaar x neue, zufällige Koordinaten generiert. So wird sichergestellt, dass ein Feld frei ist oder von nur einer Zahl bzw Linienzug belegt wird.

Nach jedem erstellten Zahlenpaar x wird ein Pfad zwischen beiden Koordinaten mittels eines Pathfinding Algorithmus generiert. Dabei wird darauf geachtet, dass der Pfad nicht über vergebene Punkte führt. Der Pfad wird dann in das Gitter eingetragen, um die verknpüfung zwischen der Zahlenpaare x_1 und x_2 sicherzustellen. So kann der Pfad zwischen x_1 und x_2 nicht von anderen Zahlen oder Pfaden überschrieben werden, wodurch die lösbarkeit des Rätsels garantiert wird. Erreicht der Pfad niemals das andere Zahlenpaar x_2, ist das Rätsel nicht lösbar, weshalb für das Zahlenpaar x neue Koordinaten generiert werden.

Nachdem alle Zahlenpaare x erstellt wurden, wird das Gitter ausgegeben.

2 Psuedocode:

Grid wird erstellt

solutions = $\frac{n}{2}$

für x $\rightarrow$ solutions

für das Zahlenpaar x (x_1 und x_2)werden zufällige x- und y-Koordinaten generiert

wenn die Koordinaten bereits vergeben sind, werden neue Koordinaten generiert

Pfad zwischen x_1 und x_2 wird generiert

wenn der Pfad von x_1 bis x_2 nicht reicht, werden neue Koordinaten generiert

Lösung und eigentliche Rätsel (nur die Zahlen im gitter ohne Pfad) werden ausgegeben - kompatibel mit dem Lösungssucher von BwInf

3 Beispiele

Ich ließ mein Programm Rätsel erstellen und im folgendem sind 2 Beispiele zu sehen, die der Lösungssucher von BwInf nicht lösen konnte und 2 Beispiele, die der Lösungssucher von BwInf lösen konnte.

3.1 Beispiel 1

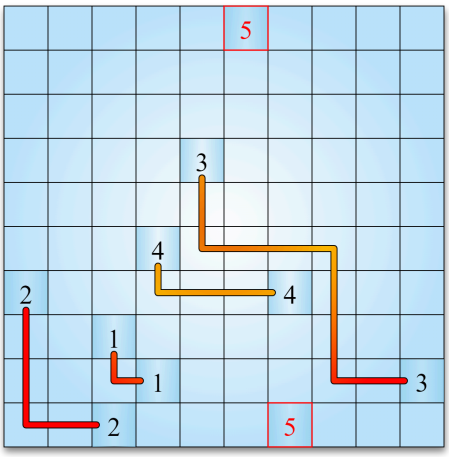

Abbildung 1:

1. Beispiel: Keine lösung vom BwInf Lösungssucher gefunden (10x10 Großes Gitter)

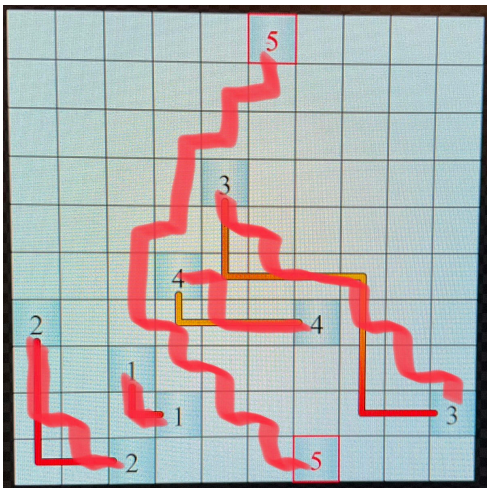

Abbildung 2:
1. Beispiel: Lösung von meinem Programm

3.2 Beispiel 2

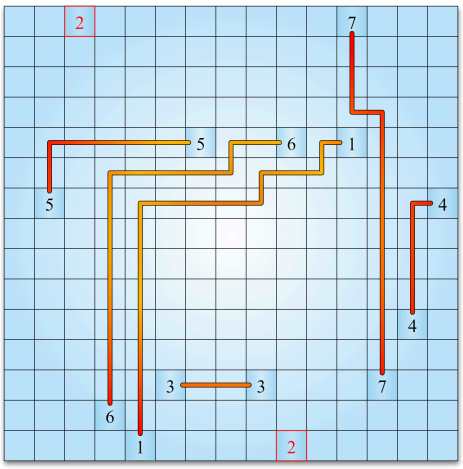

Abbildung 3:
2. Beispiel: Keine lösung vom BwInf Lösungssucher gefunden (15x15 Großes Gitter)

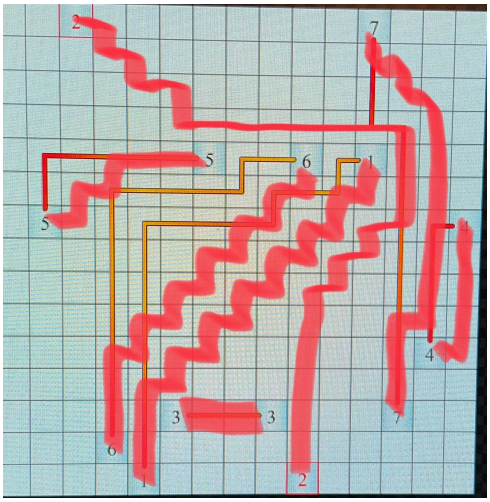

Abbildung 4:
2. Beispiel: Lösung von meinem Programm

3.3 Beispiel 3

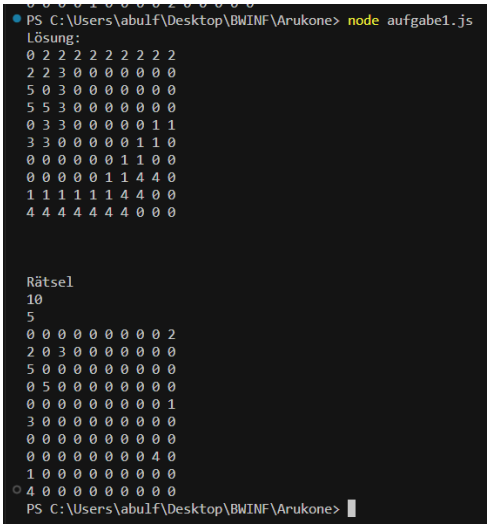

Abbildung 5:
3. Beispiel: Generiertes Rätsel von meinem Programm inklusive Lösung (10x10 Großes Gitter)

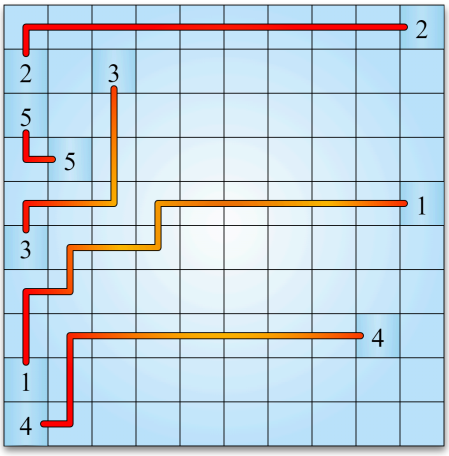

Abbildung 6:
3. Beispiel: Lösung vom BwInf Lösungssucher gefunden (10x10 Großes Gitter)

3.4 Beispiel 4

```
PS C:\Users\abulf\Desktop\BWINF\Arukone> nod
Lösung:
0 1 6 6 6 6 6 6 0 0 0 0 0 0 0
0 1 6 5 5 5 5 6 0 0 0 0 0 0 0
0 1 6 0 0 0 5 5 0 0 0 0 0 0 0
0 1 6 6 0 0 0 5 5 0 0 0 0 0 0
0 1 1 6 6 0 0 0 5 5 0 0 0 0 0
0 0 1 1 6 6 0 0 4 5 5 0 0 0 0
0 0 0 1 1 6 0 0 4 4 5 5 0 0 0
0 0 0 0 3 6 0 0 0 4 4 5 5 0 0
0 0 0 0 3 6 0 0 0 0 4 4 5 5 0
0 0 0 0 3 6 6 0 0 0 0 2 2 5 0
0 7 7 0 3 0 6 6 0 0 2 2 5 5 0
0 7 5 5 3 0 0 6 6 2 2 5 5 0 0
0 7 0 5 3 2 2 2 2 2 5 5 0 0 0
0 7 7 5 3 3 3 3 5 5 5 0 0 0 0
0 0 0 5 5 5 5 5 5 0 0 0 0 0 0



Rätsel
15
7
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 5 0 0 0 6 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 4 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 3 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 4 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 2 0 0
0 0 7 0 0 0 0 0 0 0 0 0 0 0 0
0 0 5 0 0 0 0 0 6 0 0 0 0 0 0
0 0 0 0 0 2 0 0 0 0 0 0 0 0 0
0 0 7 0 0 0 0 3 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
PS C:\Users\abulf\Desktop\BWINF\Arukone>
```

Abbildung 7:
4. Beispiel: Generiertes Rätsel von meinem Programm inklusive Lösung (15x15 Großes Gitter)

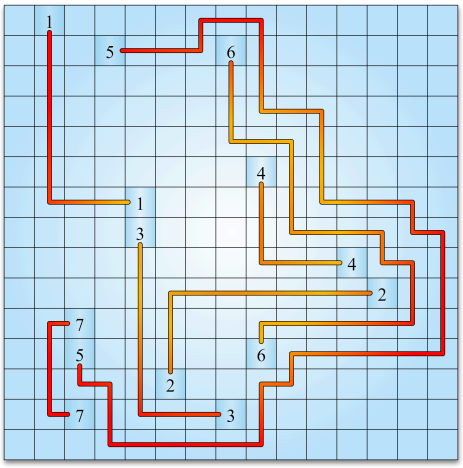

Abbildung 8:
4. Beispiel: Lösung vom BwInf Lösungssucher gefunden (15x15 Großes Gitter)

4 Quellcode

```
function arukone(n) {
    // Arukone muss >= 4 sein
```

```
    if(n < 4) {
           return console.log("Arukone muss >= 4 sein... Veränderen Sie ihre
eingabe")
    }

    // Grid erstellen
    let grid = []
    for(let i=0; i<n; i++) {
        let row = []
        for(let i=0; i<n; i++) {
            row.push(0)
        }
        grid.push(row)
    }

    // Zahlenpaare rechnen
    let solutions = Math.floor(n/2)

    // Positionen der Zahlen speichern
    let pos = {}

    function one(i) {
        let y = Math.floor(Math.random() * n)
        let x = Math.floor(Math.random() * n)
        if(grid[y][x] !== 0) { // Generierte Zahl darf nichts anderes als freie
Felder überschreiben
            return one(i);
        }
        pos[i] = {}
        pos[i][0] = y
        pos[i][1] = x
        grid[y][x] = i
        return two(i);
    }

    function two(i) {
        let y = Math.floor(Math.random() * n)
        let x = Math.floor(Math.random() * n)
        if(grid[y][x] !== 0) { // Generierte Zahl darf nichts anderes als freie
Felder überschreiben
            two(i);
            return
        }
        pos[i][2] = y
        pos[i][3] = x
        grid[y][x] = i
        return possible(i)
    }

    function possible(i) {
        // Koordinaten Punkt a
        let y = pos[i][0]
        let x = pos[i][1]

        // Koordinaten Punkt b
        let Y = pos[i][2]
        let X = pos[i][3]
```

```
grid[y][x] = 1
grid[Y][X] = 0

let check = [[y, x, 2]]
while (check.length > 0) {
    let toDelete = check.length
    for(j of check) {
        if((j[1]+1) < n) {
            if(grid[j[0]][j[1]+1] === 0) { // rechts
                grid[j[0]][j[1]+1] = j[2]
                check.push([j[0], j[1]+1, j[2]+1])
            }
        }
        if((j[1]-1) >= 0) {
            if(grid[j[0]][j[1]-1] === 0) { // links
                grid[j[0]][j[1]-1] = j[2]
                check.push([j[0], j[1]-1, j[2]+1])
            }
        }
        if((j[0]+1) < n) {
            if(grid[j[0]+1][j[1]] === 0) { // unten
                grid[j[0]+1][j[1]] = j[2]
                check.push([j[0]+1, j[1], j[2]+1])
            }
        }
        if((j[0]-1) >= 0) {
            if(grid[j[0]-1][j[1]] === 0) { // oben
                grid[j[0]-1][j[1]] = j[2]
                check.push([j[0]-1, j[1], j[2]+1])
            }
        }
    }
    check.splice(0, toDelete)
}

let temp = grid[Y][X]
if(temp === 0) {
    one(i)
}
let protection = true
while (temp > 1) {
    if((X+1) < n) {
        if(grid[Y][X+1] === temp-1) {
            grid[Y][X+1] = i
            X++
            temp--
            protection = false
        }
    }
    if((X-1) >= 0) {
        if(grid[Y][X-1] === temp-1) {
            grid[Y][X-1] = i
            X--
            temp--
            protection = false
        }
    }
    if((Y+1) < n) {
```

```
                if(grid[Y+1][X] === temp-1) {
                    grid[Y+1][X] = i
                    Y++
                    temp--
                    protection = false
                }
            }
            if((Y-1) >= 0) {
                if(grid[Y-1][X] === temp-1) {
                    grid[Y-1][X] = i
                    Y--
                    temp--
                    protection = false
                }
            }
            if(protection) {
                return one(i)
            }
            protection = true
        }

        grid[pos[i][0]][pos[i][1]] = i
        grid[pos[i][2]][pos[i][3]] = i

        // cleaning
        for(i in grid) {
            for(j in grid[i]) {
                if(typeof(grid[i][j]) === "number") {
                    grid[i][j] = 0
                }
            }
        }
    }

    // Für jedes Zahlenpaar werden die Zahlen im grid hinzugefügt und überprüft,
ob sie sich verknüpfen lassen
    for(let i=1; i<=solutions; i++) {
        one(`${i}`)
    }

    // Erstellen des finalen Rätsels
    let result = []
    for(let i=0; i<n; i++) {
        let row = []
        for(let i=0; i<n; i++) {
            row.push(0)
        }
        result.push(row)
    }

    for(let i=1; i<=solutions; i++) {
        result[pos[i][0]][pos[i][1]] = i
        result[pos[i][2]][pos[i][3]] = i
    }

    // Ausgabe der Lösungen:
    console.log("Lösung:")
    for(i of grid) {
```

```
        console.log(i.join(" "))
    }

    // Ausgabe des Rätsels
    console.log(`\n\n\nRätsel\n${n}\n${solutions}`)
    for(i of result) {
        console.log(i.join(" "))
    }
}

arukone(15)
```