Aufgabe 4: Dreieckspuzzle

Vorwort

Alle Methoden sind mit HTML und Javascript implementiert.
Das Ausführbare Programm kann man bei folgende Adresse finden:
https://tnthunter22.github.io/Bwinf_2020/Aufgabe%204/Aufgabe_4.HTML

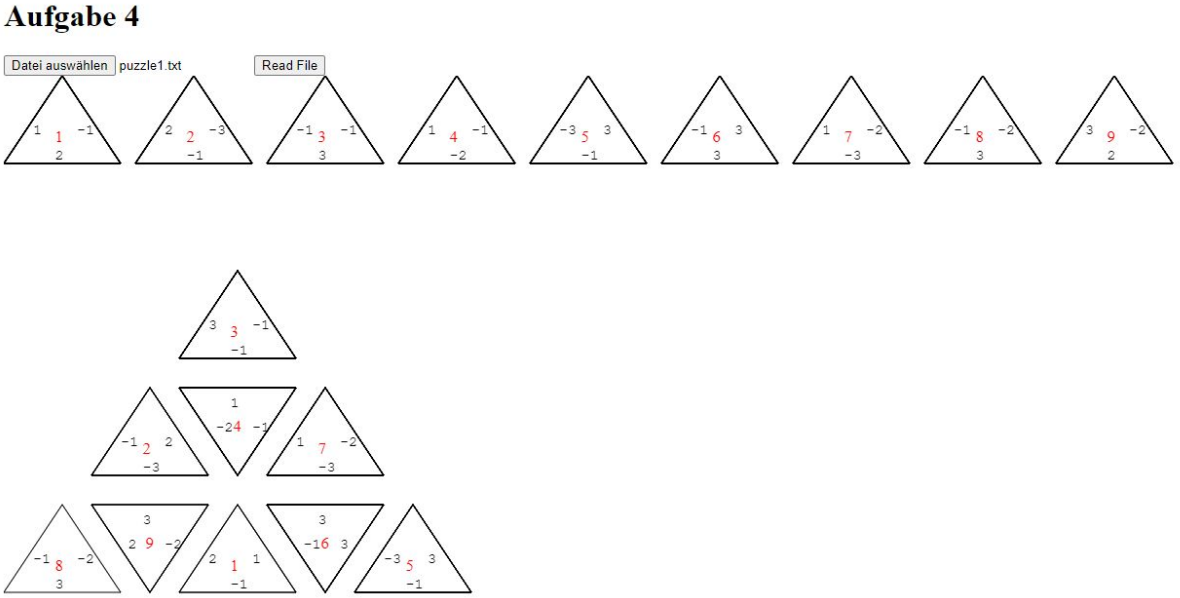
Lösungsidee

Meine Idee war ein Programm zu erstellen, dass ein Dreieck nimmt, dann schaut ob ein anderes zu diesem passt, dann schaut ob ein Dreieck zu dem passt usw. wenn keins mehr passt dann kehrt man zu dem letzten schritt zurück ,wenn es keinen schritt zurück gibt dann dreht man die Dreiecke, aber wenn aller dreiecke benutzt sind dann gibt man antwort herausgeben.

Umsetzung

Schritt 1: Eine liste mit den benutzten Dreiecken erstellen(am Anfang ist leer)
Schritt 2: Eine liste mit den unbenutzten Dreiecken anhand der benutzten Dreiecken erstellen
Schritt 3: Aus unbenutzte Dreiecken Liste passende Dreiecke für den jetzigen Platz finden
Schritt 4: In Schleife für jedes Passende Dreieck neue Liste von benutzten erstellen und rekursiv ab Schritt 1 wiederholen
Schritt 5: Ausgang aus rekursion sind: leere liste von passenden Dreiecken oder 9te rekursion(Ergebnis erfolgreich gefunden)
Schritt 6: Wenn kein ergebnis gefunden dann versucht man die Dreiecke zu drehen und ab Schritt 1 alles wiederholen

Beispiele



Quellcode

Hauptmethode die alle anderen aufruft. Hier wird auch das drehen implementiert(Schritt 6).

```
var startpuzzle = function(){
	clearCanvas();
	drawAllSmallTriangles();

	do{
		for(var n = 0;n <rotationcounter.length;n++){
			triangleList[n] = triangleRotator(rotationcounter[n],originalTriangles[n])
		}
		var path = pathFinder(0,[]);
	}while(path.length === 0 && incrementRotationcounter(0))

	if(path.length === 0){
		noResultFound();
	}else{
		drawResult(path);
	}
}
```

Rekursive Methode die Schritt 1 bis 5 ausführt

```
var pathFinder = function(n,usedTriangles){
	if(n === 9){
```

```
                return usedTriangles;
        }
        var suitableWords = []
        var unusedTriangles = getUnusedTriangles(usedTriangles);
        suitableWords = findSuitableTriangles(n,unusedTriangles,usedTriangles);

        for(var i = 0;i < suitableWords.length;i++){
                var usedTrianglesForNextStep =
getUsedTrianglesForNextStep(usedTriangles,suitableWords[i]);
                var result =  pathFinder(n + 1,usedTrianglesForNextStep)
                if (result.length > 0){
                        return result;
                }
        }
        return [];
}
```

Logik der Zählung der Drehungen

```
var incrementRotationcounter = function(index){
        if (index === 9){
                return false;
        }
        rotationcounter[index]++;
        if(rotationcounter[index] === 3){
                rotationcounter[index] = 0;
                return incrementRotationcounter(index + 1);
        }
        return true;
}
```

Die Drehung

```
var triangleRotator = function(n,triangle){
        var retval = new
Triangle(triangle.number1,triangle.number2,triangle.number3,triangle.index);;
        switch(n){
                case 1:
                retval = new
Triangle(triangle.number3,triangle.number1,triangle.number2,retval.index,);
                break;
                case 2:
                retval = new
Triangle(triangle.number2,triangle.number3,triangle.number1,retval.index);
        }
        return retval;
}
```

Methode die Schritt 3 implementiert

```
var findSuitableTriangles = function(n,unusedTriangles,usedTriangles){
	var result = [];
	for(var i = 0;i < unusedTriangles.length;i++){
		var checkresult = false;
		switch(n){
			case 0:
				checkresult = true;
				break;
			case 1:
				if(usedTriangles[0].number3 == -unusedTriangles[i].number1){
					checkresult = true;
				}
				break;
			case 2:
				if(usedTriangles[1].number2 == -unusedTriangles[i].number1){
					checkresult = true;
				}
				break;
			case 3:
				if(usedTriangles[2].number3 == -unusedTriangles[i].number1){
					checkresult = true;
				}
				break;
			case 4:
				if(usedTriangles[3].number2 == -unusedTriangles[i].number1){
					checkresult = true;
				}
				break;
			case 5:
				if(usedTriangles[1].number3 == -unusedTriangles[i].number2){
					checkresult = true;
				}
				break;
			case 6:
				if(usedTriangles[5].number3 == -unusedTriangles[i].number1){
					checkresult = true;
				}
				break;
			case 7:
				if(usedTriangles[6].number2 == -unusedTriangles[i].number1
&& usedTriangles[3].number3 == -unusedTriangles[i].number2){
					checkresult = true;
				}
				break;
			case 8:
				if(usedTriangles[6].number3 == -unusedTriangles[i].number2){
					checkresult = true;
				}
				break;
		}
		if(checkresult === true){
```

```
                        result.push(unusedTriangles[i])
                }
        }
        return result;
}
```

Methode die Schritt 2 implementiert

```
var getUnusedTriangles = function(usedTriangles){
        var unusedTriangles = [];
        mainLoop:
        for (i = 0;i < triangleList.length;i++){
                for (j = 0;j < usedTriangles.length;j++){
                        if(triangleList[i] === usedTriangles[j]){
                                continue mainLoop;
                        }
                }
        unusedTriangles[unusedTriangles.length] = triangleList[i];
        }
        return unusedTriangles;
}
```