# Java Oefeningen – Reeks 5 (extra)

## Oef 4

In dit deel gaan we algoritmes implementeren, die voor ons de puzzel kan oplossen met een optimaal aantal zetten. We gaan **breadth-first search** en **A*-search** implementeren.

Gebruik de meegegeven klasse **ZoekAlgoritmen**. Deze heeft twee (lege) methodes: **breadthfirst** en **aStar**. Gegeven een **startnode**, zal deze methode de optimale serie van moves teruggeven die de puzzel zo snel mogelijk oplost.

We moeten eerst hiervoor **PuzzelNode** het interface **ZoekNode** laten implementeren zodat deze met **ZoekAlgoritmen** kan samenwerken. Ook gaan we voor A* een "PriorityQueue" gebruiken; die heeft het **Comparable** interface nodig. We krijgen dan:

```
public class PuzzelNode implements ZoekNode<Direction>, Comparable<PuzzelNode>
```

Voeg de volgende attributen toe aan PuzzelNode: integers **g_score** en **f_score**, die de afgelegde afstand en de heuristische score bijhouden (zie slides voor meer info). Maak ook attributen van het type **Direction** en **PuzzelNode** die de vorige zet en de vorige staat van het grid bijhouden.

Maak dan nog de volgende methodes in **PuzzelNode**:

- **int distance(int a, int b):** geeft de "Manhattan"-afstand (of $\mathbb{L}_1$-afstand) terug tussen twee posities in **data**, **a** en **b**.
- **int heuristic():** berekent een ondergrens op het minimaal aantal zetten die nodig is om tot de oplossing te komen. Gebruik hiervoor de som van alle Mahalanobis-afstanden van de huidige posities van de blokjes en hun eindbestemming.
- **boolean isSolution()**: geeft terug of het probleem al dan niet opgelost is (Tip: deze methode kan heel eenvoudig geschreven worden met **heuristic()**).

- **public PuzzelNode clone():** Geeft een kopie van zichzelf terug als returnwaarde (dus met dezelfde **data**, **size**, en **zeropos**).
- **public Direction move():** Geeft de vorige zet terug.
- **public ZoekNode<Direction> move():** Geeft de vorige staat terug.
- **public void undoMove(Direction zet):** beweeg het grid in de tegengestelde richting van de meegegeven zet.

- **boolean equals(Object other):** Geeft terug of twee PuzzelNodes dezelfde **data** hebben (Tip: gebruik ***Arrays.equals***). Controleer eerst of het meegegeven object van type **PuzzelNode** is met het keyword **instanceof**. Geef anders altijd **false** terug.
- **int hashcode():** return hiervoor **Arrays.hashCode(data).** Zie slides voor uitleg.
- **public int compareTo(PuzzelNode other):** Dit wordt gebruikt voor het "Comparable" interface voor het A*-algoritme, om nodes te classificeren. Geef een int groter dan 0

als `this.f_score > other.f_score`, kleiner dan nul als `this.f_score < other.f_score` en geef 0 terug als `this.f_score == other.f_score`.

Met deze methodes kun je de zoekalgoritmen implementeren. Je vindt pseudocode op de laatste pagina van dit document. Roep breadth-first wanneer je op de **"S"** toets drukt, en A* wanneer je op de **"A"** toets drukt. Print steeds de lijst van richtingen uit. Controleer of het klopt.

## Oef 5

Tenslotte gaan we de oplossing animeren. Maak in Puzzelpaneel het attribuut: **boolean occupied = false**, die aangeeft of de code bezig is met het animeren van de oplossing zodat de animatie niet kan onderbroken worden.

Maak binnen Puzzelpaneel de methode animate_solution(List<Direction> solution). Roep deze op wanner je op de **"S"** of **"A"** toets drukt. Zet occupied op **true**, en reken de oplossing uit via **Astar**. Maak in deze methode een TimerTask object waarin je per run() een blokje verschuift. Roep de timer op met een periode van 500 ms. Cancel de Timer wanneer de oplossing bereikt is, en zet occupied weer op **false.**

Zorg tenslotte dat er niets gebeurt wanneer je op een toets drukt/klikt op het grid zolang **occupied** op **true** staat.

## Pseudocode Breadth-first search

De pseudocode voor het breadth-first search algoritme is het volgende:

```
Queue van nodes openNodes
voeg startnode toe aan openNodes

while (openNodes is niet leeg)
     haal node, het eerste element, uit openNodes

     if (node.grid is opgelost)
          return oplossing

     for (elke child uit node.possibleMoves)
          voeg child toe aan openset

(geen oplossing gevonden)
```

## Pseudocode A* search

De pseudocode voor het A* search algoritme is het volgende:

```
Gesorteerde queue van nodes openset (op basis van f_score)
Set van nodes closedset
voeg startnode toe aan openset

while (openset is niet leeg)
     haal node, het eerste element, uit openset

     if (node.grid is opgelost)
          return oplossing


     voeg node toe aan closedset

     for (elke child uit node.possibleMoves)
          if (closedset bevat child niet)
               voeg child toe aan openset

(geen oplossing gevonden)
```