Complexiteit & Algoritmiek

Backtracking

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **Joram Sjamaar** |  | **Stijn van den Berg** |  |
|  | 462398 |  | 466072 |  |

Inhoudsopgave

[Backtracking 2](#_Toc34303036)

[Ontwerp 2](#_Toc34303037)

[DirectedLine 2](#_Toc34303038)

[Maze 2](#_Toc34303039)

[Node 2](#_Toc34303040)

[Pawn 2](#_Toc34303041)

[State 2](#_Toc34303042)

[Inlezing maze 3](#_Toc34303043)

[Tests 3](#_Toc34303044)

[Resultaat 4](#_Toc34303045)

[Class Diagram 5](#_Toc34303046)

# Backtracking

In deze opdracht hebben we gewerkt met backtracking. Dit is een ‘slimme’ methode om informatie mee op te zoeken. Het is slim omdat niet elke oplossing berekend hoeft te worden.

## Ontwerp

De opdracht omschreef een doolhof met twee pionnen. Een pion mag alleen over een lijn van de kleur waar de één van de pionnen op staan.

Dit is interresant! Dat betekend dat we een aantal model classes hebben. Namelijk:

* DirectedLine
* Maze
* Node
* Pawn
* State

DirectedLine  
Heeft twee eigenschappen: Kleur en welke Node deze lijn verbind.

Maze  
Bevat alle logica en backtracking algoritme.

NodeStelt een blokje/rondje voor in het doolhof. Heeft twee eigenschappen: Een nummer en kleur.

Pawn  
Stelt een pion voor. Deze class is niet nodig. Ik heb toch voor deze class gekozen omdat het mij duidelijkheid gaf. De pion houd alleen bij op welke Node deze staat.

State  
Houd de toestand bij van het doolhof.  
  
De meest makkelijke vergelijking die ik kan maken, is een foto. Stel je voor dat je een foto maakt van een schaakspel. Je ziet dan precies waar alle spelstukken staan. Dit doet de State ook. Het slaat op waar de pionnen staan.

## Ondersteuning voor meerdere doolhoffen

De ondersteuning voor meerdere doolhoffen wordt gerealiseerd met een JSON inlezing van een doolhof.

De maze wordt standaard ingelezen vanuit het *maze.json* bestand.

Dit JSON bestand bevat een array, met daarin Nodes en de daarbij behoordende properties

**Specificatie**

Array

* Object *(Node)*
  + Number *(int)*
  + Color *(“PINK” | “GREEN” | “BLACK” | “ORANGE” | “BLUE”)*
  + Lines *(Array)*
    - Object
      * Color *(“PINK” | “GREEN” | “BLACK” | “ORANGE” | “BLUE”)*
      * pointsTo *(int)*

**Voorbeeld**

[  
 {  
 "number": 1,  
 "color": "PINK",  
 "lines": [  
 {  
 "color": "PINK",  
 "pointsTo": 4  
 },  
 {  
 "color": "BLACK",  
 "pointsTo": 5  
 }  
 ]  
 }

]

## Tests

Er zijn twee tests. En hier is iets mee mis.

Als er een andere test runt, dan crasht de eerste test *(totalStepsShouldBe37)*

Mocht u zo vriendelijk willen zijn, om de andere test dan te commenten.

# Resultaat

Het resultaat is 37 stappen voordat de oplossing gevonden is.

Als je de states van het resultaat na loopt dan klopt dit.

De recursieve functie werkt dus tot behoren.

Met de JSON import is er ook mogelijkheid tot meerdere mazes, of kleine veranderingen in mazes. Hierdoor kun je de andere resultaten (of geen resultaten) ook bekijken.

# Class Diagram

