Better Code Hub (BCH) evaluatie

Louis Matthijssen – ITV2D – Opdracht “Boggle”

In dit document zal ik kort de beoordeling van Better Code Hub voor de opdracht “Boggle” evalueren. Ik zal elk onderdeel van de BCH-resultaten apart behandelen. Als laatste zal ik ook de complexiteit van de algoritmen bespreken.

# ✔ Write Short Units of Code

* De “SolveService.solve” methode is 1 regel te lang (16 regels). Aangezien Deze methode lastig op te splitsen is (er wordt al maar 1 taak uitgevoerd) en de methode ook een paar log regels bevat, zou ik deze methode niet opsplitsen.

# ✔ Write Simple Units of Code

BCH heeft bij dit onderdeel geen opmerkingen.

# ✔ Write Code Once

BCH heeft bij dit onderdeel geen opmerkingen.

# ✔ Keep Unit Interfaces Small

* De “SolveWorker.visitPoint”, “LegendDisplay.drawLegendColor”, “BoardDisplay.drawPoint” en “LegendDisplay.syncColorWithLabel” methodes hebben allemaal 1 parameter te veel (3 parameters). Bij al deze methodes heb ik al gebruik gemaakt van models om het aantal parameters te verkleinen. Aangezien er bij deze methodes maar 1 parameter te veel is zou ik dit niet aanpassen.
* De “BoardModel.addPointToList” methode heeft 1 parameter te veel (3 parameters). Aangezien dit een redelijk simpele functie is zou ik hier niet een apart model voor aanmaken.
* De “SolveWorkerDataModel.$constructor” en “MatchModel.$constructor” hebben allebei 1 parameter te veel (3 parameters). Aangezien het hier gaat om de constructor van models lijkt mij dit geen probleem.

# ✔ Separate Concerns in Modules

* De “DataManager” wordt op 1 plek te veel aangeroepen (11 plekken). Deze klasse laat controllers data met elkaar delen. Hier is misschien een betere oplossing voor maar aangezien het een klein probleem is zou ik dit zo laten.

# ✔ Couple Architecture Components Loosely

BCH heeft bij dit onderdeel geen opmerkingen.

# ✘ Keep Architecture Components Balanced

* BCH raadt aan om minstens 2 componenten te gebruiken, deze applicatie heeft er 1. Het is mogelijk om de GUI en het algoritme zelf op te splitsen, maar gezien de grootte van de applicatie lijkt mij dit niet nodig.

# ✔ Keep Your Codebase Small

BCH heeft bij dit onderdeel geen opmerkingen.

# ✔ Automate Tests

BCH heeft bij dit onderdeel geen opmerkingen.

# ✔ Write Clean Code

BCH heeft bij dit onderdeel geen opmerkingen.

# Complexiteit van de algoritmen

Het gebruikte algoritme is redelijk complex. Dit komt omdat er woorden in alle mogelijke richtingen gevonden kunnen worden.

De onderstaande stappen beschrijven de basis van het gebruikte algoritme:

1. Het algoritme moet gestart worden vanaf elk punt op het bord. Bij een bord van 3x3 wordt het algoritme dus 9 keer uitgevoerd. Op deze manier wordt er vanaf elk punt gekeken welke woorden er gevormd kunnen worden.
2. De letter van het opgegeven punt wordt opgehaald en toegevoegd aan de combinatie van letters in het huidige pad. Hierna wordt het punt toegevoegd aan de lijst met bezochte punten.
3. Er wordt gekeken of de huidige combinatie van letters overeenkomt met 1 van de gezochte woorden. Als dit het geval is worden de bezochte punten en het gevonden woord opgeslagen en gaat het algoritme verder om eventueel langere woorden te vinden.
4. Alle omliggende punten van het huidige punt worden opgehaald. Voor elk omliggende punt wordt gekeken of deze in de lijst met bezochte punten voorkomt. Is dit het geval, dan wordt dit punt overgeslagen (een punt mag maar 1 keer bezocht worden). Anders wordt de methode uit stap 2 opnieuw aangeroepen met de combinatie van letters in het huidige pad, de lijst met bezochte punten en het nieuwe (omliggende) punt dat bezocht moet worden.

Helaas kost het bovenstaande algoritme veel rekenkracht. Voor een bord van 3x3 zou dit 10.305 stappen kosten, en voor een bord van 4x4 zouden dit al 12.029.640 stappen zijn. Hierdoor duurt het oplossen van een bord steeds langer naarmate het bord groter wordt.

Om dit op te lossen heb ik nog een simpele base case toegevoegd:

* Alle delen van alle woorden in de lijst met te zoeken woorden worden voor het starten van het algoritme opgeslagen in een HashSet. Voor het woord “BOUW” worden bijvoorbeeld “B”, “BO” en “BOU” opgeslagen. Er wordt hier gebruik gemaakt van een HashSet omdat het zoeken hierin een O(1) operatie is en deze HashSet bij elke stap in het algoritme gebruikt zal worden.
* Bij elke stap van het bovenstaande algoritme wordt gekeken of de huidige combinatie van letters voorkomt in deze HashSet. Is dit niet het geval, dan is het niet meer mogelijk om een woord te vinden met de huidige combinatie van letters en kan het huidige pad afgebroken worden.

Deze base case verlaagt het aantal stappen flink. Het aantal stappen is nu wel afhankelijk van de grootte van de lijst met te zoeken woorden. Enkele voorbeelden (met een lijst van 11.201 woorden van Blackboard):

* 3x3 bord: 147 stappen
* 4x4 bord: 342 stappen

Hiermee is de complexiteit van dit algoritme , waarbij:

* N de grootte van het bord is (bij een bord van 3x3 is dit 3)
  + Met wordt dit het totale aantal vakjes/beginpunten
* d de maximale diepte van een pad is (in dit geval de lengte van het langste te vinden woord)
  + 8 is hierbij het aantal omliggende vakjes per vakje, is de complexiteit van het depth-first search algoritme, wat hier is toegepast (zie Wikipedia: <https://en.wikipedia.org/wiki/Depth-first_search>).