Opdracht 1, Programmeertechnieken

Jenny Vermeltfoort, s3787494, groep 48

31 maart 2025

1 Uitleg

Dit verslag omschrijft gedeeltelijk een implementatie van een *engine* die het spel aqualin speelt. Aqualin is een spel waar twee speler het tegen elkaar opnemen. Het vind plaats op een 2d gerasteriseerd bord. Spelers leggen stenen op het bord; de speler met het grootste cluster aan het eind van het spel heeft gewonnen. Stenen bestaan uit een vorm en een kleur, het is dus de bedoeling dat de ene speler clusters maakt van kleuren en de andere speler clusters van vormen. Stenen worden op een bepaalde volgorde op het bord gelegd, het is dus aan de speler om te bepalen welke steen het beste past op de te leggen positie. Het spel heeft een aantal parameters, de maximaal hoeveelheid stenen per hand – wat per speler kan verschillen, de grootte van het bord, de hoeveelheid vormen, en hoeveel stenen er in het begin vanuit de pot op het bord worden gelegd. De implementatie bestaat uit een aantal methodes om te bepalen welke steen het beste gelegd kan worden voor een bepaalde stand van het bord.

1.1 Optimale score.

De implementatie bepaald de eindscore door alle leaves van de actie-toestand boom te bekijken. Voor elke stand bij de leaves word de score berekend, bij Brian als initiële speler: (grooste_cluster_brian - grootste_cluster_robin). Door voor elke zet de score te vergelijken is het mogelijk om de optimale score te bepalen en daarmee de optimale zet.

De implementatie maakt gebruik van doeZet/unDoeZet. Een kopie maken van het bord is onnodig, aangezien de staat van het bord niet wordt veranderd wanneer er een doeZet vervolgd met een unDoeZet uitgevoerd word. Er wordt namelijk in de hand van elke speler bijgehouden op welke positie elke unieke steen komt te zitten. Bijvoorbeeld wanneer steen (1,2) op de tweede positie uit de hand van Brian wordt gehaald bij een doeZet, dan komt deze later ook weer terug op de twee positie bij een unDoeZet.

De toestands-actie boom eerder afkappen lijkt niet mogelijk. Vandaar dat mijn implementatie de volledige boom afloopt. Een aantal dingen die zijn overwogen:

1. Een hoogste score bijhouden voor elke node en wanneer deze score niet meer gehaald kan worden de node in de boom overslaan. Dit is wanneer de hoeveelheid te maken standen plus de huidige grootste cluster minder is dan deze perfecte score. Deze beredenering is echter incorrect, aangezien met de laatste zet vier clusters met elkaar verbonden kunnen worden. Een voorbeeld is wanneer vier clusters van grootte twee met elkaar worden verbonden met de laatste zet, dan gaat de grootste cluster voor de speler plots van twee naar acht. Dit is dus alleen te bepalen door de volledige boom af te lopen.

2. Bestaat er symmetrie in de boom? Nee, aangezien elke zet die gemaakt kan worden produceert een tot een uniek bord, elke stand van het bord bij de leaves van de toestand-actie boom is dus uniek.

3. Voor elke speler een hash map bijhouden die de staat van het bord en de staat van de hand van de speler hash'd en mapped aan de te behalen scoren; voor elke bezochte node deze hash map vullen met deze score en met de hash map kijken of de hash voor de huidige stand van het bord al eens bepaald is. Zo ja, hoeft de hele tak van node tot leave niet opnieuw bepaald te worden; je kan namelijk gewoon de hash map uitlezen. Dit zou theoretisch mogelijk zijn, echter wordt deze hash map ontzettend groot, deze heeft namelijk dezelfde grootte als de hoeveelheid leaves van de toestands-actie boom.

4. Een maal de optionele score berekenen door de hele boom af te lopen en vervolgens deze score cachen. Wanneer de optionele score opnieuw wordt berekend voor nodes verder dan de gecachete nodes, dan kunnen alle leaves met deze score vergeleken worden. Wanneer deze hoogste score behaald kan worden met de huidige tak van de boom dan hoeft de rest van de toestands-actie boom niet meer worden afgelopen; je hebt namelijk al de zetten gevonden om de optionele score te behalen. Dit is echter niet geïmplementeert in mijn implementatie.

2 Spellen

In de volgende subsecties zijn de resultaten van de implementatie te zien. In de resultaten is dat de Monte Carlo implementatie vergelijkbare resultaten bied als het berekenen van de optimale score. Dit is interessant, aangezien de Monte Carlo methode altijd maar 100 * k0 of 100 * k1 aantal standen bekijkt. Dit is dus in vergelijking met de optimale score berekening aanzienlijk sneller, bij spel1 een factor 80k sneller.

De grootste cluster implementatie presteert echter ondermaats, maar is daarentegen wel ontzettend snel.

2.1 Spel 1

Resultaten voor spel1.txt

```
    Optimale score is: 0
    Een optimale zet is: (1,2)
    We hebben hiervoor 16777216 standen bekeken.
    Dit kostte 6361731 clock ticks, ofwel 6.36173 seconden.

Score grootste cluster tegen optimaal is: -3

Score Monte Carlo tegen optimaal is: 0
```

2.2 Spel 2

Resultaten voor spel2.txt

```
    Optimale score is: 2
    Een optimale zet is: (2,0)
    We hebben hiervoor 93312 standen bekeken.
    Dit kostte 19915 clock ticks, ofwel 0.019915 seconden.
```

```
Score grootste cluster tegen optimaal is: 1

Score Monte Carlo tegen optimaal is: 2
```

3 Toestand-actie boom

De hoeveelheid leaves van de toestands-actie boom is te berekenen met de volgende formule:

$$l = \frac{k_0 * \lceil p/2 \rceil * k_1 * \lfloor p/2 \rfloor * k_0! * k_1!}{n - f(k_0, k_1, p)} \tag{1}$$

De factor $\frac{1}{n-m}$ is een correctie voor wanneer het spel eerder stopt. Dit gebeurd wanneer een van de spelers geen stenen meer heeft, dit komt voor wanneer het verschil $|k_0 - k_1| >= 3$.

Waarbij n de hoeveelheid te spelen stenen.

$$n = hoogte * breedte - alopbord \tag{2}$$

De variabele p is de hoeveelheid te leggen stenen van begin toestand tot eind toestand.

$$p = n - k_0 - k_1 \tag{3}$$

De variabele m is de hoeveelheid standen. Hierbij floor en ceil omdat Brian een steen meer krijg als Robin wanneer p oneven is. En $+1$ in de twee lijn, omdat Brian een zet meer krijgt wanneer Robin's hand groter is.

$$f(k_0, k_1, p) = \begin{cases} 2k_0 + \lceil \frac{p}{2} \rceil \cdot 2 & \text{als } k_0 < k_1 \\ 2k_1 + \lfloor \frac{p}{2} \rfloor \cdot 2 + 1 & \text{als } k_0 > k_1 \\ k_0 + k_1 + p & \text{als } k_0 = k_1 \end{cases} \tag{4}$$

4 Grootste Cluster

Zie onderstaande tabel en het stuk tekst onder de tabel waarin de handen van de spelers wordt weergegeven. De speler met de vormen is aan de beurt. In deze toestand zal de Grootste Cluster implementatie ervoor kiezen om steen (3,4) op positie 1 te leggen, leidende tot een grootste cluster van drie stenen. Echter leidt hier opeenvolgend steen (1,0) op positie 1 of 3 en steen (5,0) op positie 1 of 3 tot een grootste cluster van vijf stenen op. Deze situatie toont aan dat het Grootste Cluster fouten maakt, omdat het niet verder kijkt dan dat haar virtuele neus lang is.

(0,1)	5	(3,5)	(2,1)	(1,3)	(5,5)
(4,3)	(2,4)	4	(0,3)	(1,4)	(0,5)
(0,2)	(3,2)	(5,3)	(5,4)	1	(0,0)
(4,2)	2	(2,3)	(3,0)	3	(2,2)
(2,0)	(4,5)	(2,5)	(4,0)	(1,2)	(0,4)
(3,1)	(1,5)	(1,1)	(5,1)	(3,3)	(5,2)

```
kleuren: (4,1), (4,4)
vormen: (1,0), (5,0), (3,4)
```

5 Experimenten

In de volgende subsecties volgende de resultaten van de experimenten. De experimenten laten zien dat de verschillende parameters van de configuraties ervoor zorgen dat de beginnende speler gemiddeld er of beter of slechter vanaf is. Dit aangezien bij zowel experiment 1 als bij experiment 2 Brian de winnende speler is. De grootte van de hand heeft invloed op de einscore, hoe groter de hand hoe hoger de kans is dat de speler wint. Dit is enigszins logisch, aangezien de speler met de grotere hand meer zetten kan analyzeren.

Verder is er te betogen dat de Monte Carlo implementatie beter een goede zet bepaald in vergelijking met de Grootste Cluster implementatie. Te zien aan experiment 2 waar Brian bij elke configuratie net iets meer scoort.

5.1 Experiment 1: GrootsteCluster vs Optimaal

Tabel 1: Samengevoegde Scores, Tijdsduur en Configuratie-informatie

Configuratie	Hoogte	Breedte	Vormen	k0	k1	Gemiddelde Score	Tijdsduur (seconden)
1	4	4	4	2	2	-0.99	0.200511
2	4	5	4	2	2	-1.69	3.56373
3	3	5	5	2	2	0.69	0.1002
4	4	4	4	3	2	-0.18	1.67778

5.2 Experiment 2: MonteCarlo vs Optimaal

Tabel 2: Samengevoegde Scores, Tijdsduur en Configuratie-informatie

Configuratie	Hoogte	Breedte	Vormen	k0	k1	Gemiddelde Score	Tijdsduur (seconden)
1	4	4	4	2	2	-0.37	0.259286
2	4	5	4	2	2	-1.03	3.69457
3	3	5	5	2	2	1.19	0.146323
4	4	4	4	3	2	0.58	1.80519

5.3 Experiment 3: GrootsteCluster vs MonteCarlo

De gemiddelde score voor dit experiment lijk enigszins instabiel; bij 100 iteraties varieert het tussen de gemiddelde score tussen -1.3 en -1.77. Met wat natte vinger werk is dit wel te verklaren, aangezien het bord bij deze configuratie de variantie aan beste zetten varieert door de grootte van het bord. Vandaar is er ook het resultaat met 1000 iteraties toegevoegd.

Tabel 3: Samengevoegde Scores, Tijdsduur en Configuratie-informatie

Iteraties	Hoogte	Breedte	Vormen	k0	k1	Gemiddelde Score	Tijdsduur (seconden)
100	10	10	10	5	5	-1.77	5.35924
1000	10	10	10	5	5	-1.604	52.5195

Appendices

A Experimenten Logs

A.1 spel1.txt

```
Optimale score is: 0
Een optimale zet is: (1,2)
We hebben hiervoor 16777216 standen bekeken.
Dit kostte 4790511 clock ticks, ofwel 4.79051 seconden.
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (-1,-1) (0,3) (0,1)  (3,4)  (-1,-1)
(-1,-1) (-1,-1) (1,3) (4,3)  (2,1)  (-1,-1)
(-1,-1) (-1,-1) (2,4) (-1,-1) (-1,-1) (-1,-1)
(5,5)  (-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (4,4)  (-1,-1) (-1,-1) (3,5) (-1,-1)
vulvolgorde (x,y):(2,5)(2,0)(1,0)(5,1)(0,2)(4,4)(3,0)(0,3)(5,4)(0,0)(1,4)(4,0)(0,1)(5,0)(3,5)
    (3,4)(5,3)(4,3)(2,4)(3,3)(1,2)(1,1)(5,5)(1,3)(5,2)(0,5)
pot (kleur,vorm):(5,3)(5,4)(1,0)(0,0)(4,2)(4,1)(2,3)(3,0)(5,0)(2,2)(2,0)(4,5)(2,5)(4,0)(1,4)
    (0,4)(3,1)(1,5)(1,1)(5,1)(3,3)(5,2)
hand_kleuren(kleur,vorm):(8: 1,2)(5: 0,5)
hand_vormen(kleur,vorm):(2: 0,2)(20: 3,2)

Score grootste cluster tegen optimaal is: -3
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (-1,-1) (0,3) (0,1)  (3,4)  (-1,-1)
(-1,-1) (-1,-1) (1,3) (4,3)  (2,1)  (-1,-1)
(-1,-1) (-1,-1) (2,4) (-1,-1) (-1,-1) (-1,-1)
(5,5)  (-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (4,4)  (-1,-1) (-1,-1) (3,5) (-1,-1)
vulvolgorde (x,y):(2,5)(2,0)(1,0)(5,1)(0,2)(4,4)(3,0)(0,3)(5,4)(0,0)(1,4)(4,0)(0,1)(5,0)(3,5)
    (3,4)(5,3)(4,3)(2,4)(3,3)(1,2)(1,1)(5,5)(1,3)(5,2)(0,5)
pot (kleur,vorm):(5,3)(5,4)(1,0)(0,0)(4,2)(4,1)(2,3)(3,0)(5,0)(2,2)(2,0)(4,5)(2,5)(4,0)(1,4)
    (0,4)(3,1)(1,5)(1,1)(5,1)(3,3)(5,2)
hand_kleuren(kleur,vorm):(8: 1,2)(5: 0,5)
hand_vormen(kleur,vorm):(2: 0,2)(20: 3,2)

Score Monte Carlo tegen optimaal is: 0
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (-1,-1) (0,3) (0,1)  (3,4)  (-1,-1)
(-1,-1) (-1,-1) (1,3) (4,3)  (2,1)  (-1,-1)
(-1,-1) (-1,-1) (2,4) (-1,-1) (-1,-1) (-1,-1)
(5,5)  (-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (4,4)  (-1,-1) (-1,-1) (3,5) (-1,-1)
```

```
vulvolgorde (x,y):(2,5)(2,0)(1,0)(5,1)(0,2)(4,4)(3,0)(0,3)(5,4)(0,0)(1,4)(4,0)(0,1)(5,0)(3,5)
    (3,4)(5,3)(4,3)(2,4)(3,3)(1,2)(1,1)(5,5)(1,3)(5,2)(0,5)
pot (kleur,vorm):(5,3)(5,4)(1,0)(0,0)(4,2)(4,1)(2,3)(3,0)(5,0)(2,2)(2,0)(4,5)(2,5)(4,0)(1,4)
    (0,4)(3,1)(1,5)(1,1)(5,1)(3,3)(5,2)
hand_kleuren(kleur,vorm):(8: 1,2)(5: 0,5)
hand_vormen(kleur,vorm):(2: 0,2)(20: 3,2)

Score grootste cluster tegen Monte Carlo is: -3
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (-1,-1) (0,3) (0,1)  (3,4)  (-1,-1)
(-1,-1) (-1,-1) (1,3) (4,3)  (2,1)  (-1,-1)
(-1,-1) (-1,-1) (2,4) (-1,-1) (-1,-1) (-1,-1)
(5,5)  (-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (4,4)  (-1,-1) (-1,-1) (3,5) (-1,-1)
vulvolgorde (x,y):(2,5)(2,0)(1,0)(5,1)(0,2)(4,4)(3,0)(0,3)(5,4)(0,0)(1,4)(4,0)(0,1)(5,0)(3,5)
    (3,4)(5,3)(4,3)(2,4)(3,3)(1,2)(1,1)(5,5)(1,3)(5,2)(0,5)
pot (kleur,vorm):(5,3)(5,4)(1,0)(0,0)(4,2)(4,1)(2,3)(3,0)(5,0)(2,2)(2,0)(4,5)(2,5)(4,0)(1,4)
    (0,4)(3,1)(1,5)(1,1)(5,1)(3,3)(5,2)
hand_kleuren(kleur,vorm):(8: 1,2)(5: 0,5)
hand_vormen(kleur,vorm):(2: 0,2)(20: 3,2)
```

A.2 spel2.txt

```
Optimale score is: 2
Een optimale zet is: (2,0)
We hebben hiervoor 93312 standen bekeken.
Dit kostte 16583 clock ticks, ofwel 0.016583 seconden.
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (-1,-1) (-1,-1) (2,1) (2,2)
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (1,1)
(-1,-1) (-1,-1) (0,5) (2,4)  (-1,-1)
vulvolgorde (x,y):(1,1)(0,1)(0,3)(1,0)(2,1)(0,2)(0,0)(4,0)(2,0)(3,0)(1,2)(3,2)(1,3)(4,3)
    (2,2)
pot (kleur,vorm):(2,5)(0,3)(3,1)(3,0)(1,0)(0,0)(0,1)(1,5)(1,2)(1,4)
hand_kleuren(kleur,vorm):(4: 0,4)(12: 2,0)
hand_vormen(kleur,vorm):(2: 0,2)(15: 2,3)(9: 1,3)

Score grootste cluster tegen optimaal is: 1
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (-1,-1) (-1,-1) (2,1) (2,2)
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (1,1)
(-1,-1) (-1,-1) (0,5) (2,4)  (-1,-1)
vulvolgorde (x,y):(1,1)(0,1)(0,3)(1,0)(2,1)(0,2)(0,0)(4,0)(2,0)(3,0)(1,2)(3,2)(1,3)(4,3)
    (2,2)
pot (kleur,vorm):(2,5)(0,3)(3,1)(3,0)(1,0)(0,0)(0,1)(1,5)(1,2)(1,4)
hand_kleuren(kleur,vorm):(4: 0,4)(12: 2,0)
hand_vormen(kleur,vorm):(2: 0,2)(15: 2,3)(9: 1,3)

Score Monte Carlo tegen optimaal is: 2
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (-1,-1) (-1,-1) (2,1) (2,2)
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (1,1)
(-1,-1) (-1,-1) (0,5) (2,4)  (-1,-1)
```

```
vulvolgorde (x,y):(1,1)(0,1)(0,3)(1,0)(2,1)(0,2)(0,0)(4,0)(2,0)(3,0)(1,2)(3,2)(1,3)(4,3)
    (2,2)
pot (kleur,vorm):(2,5)(0,3)(3,1)(3,0)(1,0)(0,0)(0,1)(1,5)(1,2)(1,4)
hand_kleuren(kleur,vorm):(4: 0,4)(12: 2,0)
hand_vormen(kleur,vorm):(2: 0,2)(15: 2,3)(9: 1,3)

Score grootste cluster tegen Monte Carlo is: 1
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (-1,-1)
(-1,-1) (-1,-1) (-1,-1) (2,1) (2,2)
(-1,-1) (-1,-1) (-1,-1) (-1,-1) (1,1)
(-1,-1) (-1,-1) (0,5) (2,4)  (-1,-1)
vulvolgorde (x,y):(1,1)(0,1)(0,3)(1,0)(2,1)(0,2)(0,0)(4,0)(2,0)(3,0)(1,2)(3,2)(1,3)(4,3)
    (2,2)
pot (kleur,vorm):(2,5)(0,3)(3,1)(3,0)(1,0)(0,0)(0,1)(1,5)(1,2)(1,4)
hand_kleuren(kleur,vorm):(4: 0,4)(12: 2,0)
hand_vormen(kleur,vorm):(2: 0,2)(15: 2,3)(9: 1,3)
```

A.3 Experiment 1

```
config 0: De gemiddelde score van Brian is: -0.99
Dit kostte 200511 clock ticks, ofwel 0.200511 seconden.

config 1: De gemiddelde score van Brian is: -1.69
Dit kostte 3563727 clock ticks, ofwel 3.56373 seconden.

config 2: De gemiddelde score van Brian is: 0.69
Dit kostte 100200 clock ticks, ofwel 0.1002 seconden.

config 3: De gemiddelde score van Brian is: -0.18
Dit kostte 1677776 clock ticks, ofwel 1.67778 seconden.
```

A.4 Experiment 2

```
config 0: De gemiddelde score van Brian is: -0.37
Dit kostte 259286 clock ticks, ofwel 0.259286 seconden.

config 1: De gemiddelde score van Brian is: -1.03
Dit kostte 3694570 clock ticks, ofwel 3.69457 seconden.

config 2: De gemiddelde score van Brian is: 1.19
Dit kostte 146323 clock ticks, ofwel 0.146323 seconden.

config 3: De gemiddelde score van Brian is: 0.58
Dit kostte 1805194 clock ticks, ofwel 1.80519 seconden.
```

A.5 Experiment 3

```
    De gemiddelde score van Brian is: -1.77
Dit kostte 5359240 clock ticks, ofwel 5.35924 seconden.
```

```
    De gemiddelde score van Brian is: -1.604
Dit kostte 52519495 clock ticks, ofwel 52.5195 seconden.
```