verslag programmeren

Isabel Baas, Daan Braakman & Mylan Koobs

January 2023

1 inleiding

2 Algoritme

2.1 Data-structuur

In de code zitten meerdere klassen. De meest algemene klasse, ..., heeft een rooster met een zelf te kiezen aantal cellen in iedere dimensie. De dimensie is ook zelf te kiezen. Voor het rooster is er gebruik gemaakt van numpy arrays. Deze maken het makkelijker om te werken met meerdere dimensies.
Daarnaast heeft de klasse een regel-functie, die zelf gedefineerd kan worden, bepaald de toestanden van de volgende generaties cellen.
Ook bevat de klasse een functie voor het updaten, deze wordt later verder toegelicht.

Er zijn twee klassen afgeleid van de algemene klasse, die voor één- en voor tweedimensionale CA's. Deze bevatten alles van de algemene klasse en dat wat er nodig is om ze grafisch weer te geven. Er is een neighborhood klasse met functies waarmee de staten van omliggende cellen van een cel mee kunnen worden opgehaald. Deze kunnen gebruikt worden bij het defineren van de regel-functies.

2.2 Update

De update-functie update het gehele rooster door voor elke cel de rules functie op te roepen om te bepalen wat zijn nieuwe staat moet zijn. De complextiteit hangt af van de complextiteit van de regel-functie. Deze noemen we m. Bij n cellen wordt de rules-functie n keer aangeroepen, dus dan is de complextiteit mn.

2.3 Grafische weergave

Voor het grootste deel zijn de draw-functies voor 1D en 2D hetzelfde. De complextiteit is bijvoorbeeld helemaal gelijk. Deze is compleet afhankelijk van het aantal cellen. Zij dat aantal gelijk aan n, dan is de complexiteit dus ook n.

Voor de grafische weergave van de ééndimensionale CA was het nodig om als het waren mee te scrollen met de nieuwe generaties. Deze raakten namelijk na een aantal generaties buiten beeld. Om dit probleem op te lossen, wordt elke nieuwe generatie toegevoegd aan een lijst. Zodra de lijst meer generaties bevat dan grafisch weer te geven zijn op het scherm, wordt de oudste generaties uit de lijst gehaald en is deze dus ook niet meer te zien.

2.4 Neighborhood

The neighboorhoud functies zijn bedoeld om bij het maken van regels-functies makkelijk toegang te krijgen tot de toestanden van buurtcellen. Er wordt gebruikt gemaakt van een Moore-Neighborhood met een zelf te kiezen afstand, die we r noemen. Ook kan dit voor elk aantal dimensies, die we d noemen (de algemene functie maakt wel gebruik van recursie en is daarom wat minder snel, dus voor 1- en 2-dimensionale CA's kan beter gebruik worden gemaakt van de daarvoor bestemde neighborhood-functies).

De functies lezen de toestanden van in totaal r^d cellen, en aangezien het lezen in constante tijd kan, is de uiteindelijke complexiteit ook r^d.

Voor de periodieke neighborhood-functie wordt nog gebruik gemaakt van modulo-operaties (modulo scherm-hoogte of -breedte), dus bij grote schermgroottes kan dit voor extra complexiteit zorgen doordat de complexiteit van modulo-operaties dan groter wordt.

2.5 Overig

Als laatste

3 handleiding

4 Iets van een discussie

4.1 samenvatting

4.2 Tekortkomingen code

Iets wat op ons opviel tijdens het afronden van de eendimensionale klasse is dat het "tekenen" van iedere nieuwe rij in de grafische weergaven steeds langzamer verliep naarmate het programma langer had gerund. Na het controleren bleek python ook een onproportioneel groot deel van de processor te gebruiken. Dit is een probleem waar we tegenaan liepen. Het probleem viel ons op toen we de code hadden toegevoegd die er voor zorgt dat wanneer de grote van de CA groter was dan het scherm, de grafische weergave wel in het midden van het scherm begon en dat de CA als het ware buiten het scherm verder loopt. Echter leek dit niet de oorzaak te zijn.
Een vergelijkbaar probleem kwam omhoog nadat de twee dimensionale klasse af was. Een CA met de regels van Game of life en een rooster grote van bijvoorbeeld 50 bij 50 gaat zo snel als is ingesteld in de *changetime* optie. Maar hoe groter het rooster wordt ingesteld, hoe langzamer de CA gaat. Er is geen directe oorzaak gevonden, niet iets kleins wat opgeglost zou kunnen worden zonder dat een groot deel van de code herschreven zou moeten worden. Aangezien we dit probleem pas laat tegenkwamen is dit ook niet opgelost.

De volgende punten zijn niet zo zeer tekortkomingen, maar meer toevoegingen die we graag hadden gedaan.
De eerste is de mogelijkheid om als het ware over het rooster heen te kunnen bewegen waneer het rooster groter is dan het scherm. Aangezien dit niet een vereiste was voor de opdracht en het waarschijnlijk erg veel werk zou vergen, hebben we besloten dit achterwegen te laten.
Daarnaast wilde we het graag mogelijk maken om cellen tot leven te brengen door er op te klikken, naast dat je geselecteerde cellen een toestand kan geven met de *setcells* optie. Zo zou het veel makkelijker zijn om bijvoorbeeld een glider gun te maken in Game of life. Het begin van het maken van deze optie is de mogelijk om de CA op pauze te zetten.

4.3 Taak verdeling & tijdsmanagement

De taakverdeling was iets dat heel natuurlijk verliep. Buiten de werkcolleges werd er appart gewerkt aan onderdelen van de code en het verslag en in de werkcolleges werkte we veel samen. Vaak dachten we samen aan oplossingen voor een onder deel wat op dat moment relevant was en typte een persoon het uit in code. De onderdelen van het verslag zijn iets bewuster verdeeld, maar ook dit ging heel simpel. Uit eindelijk is al het werk dat nodig was voor het project niet compleet evenredig verdeeld, maar niet in mate dat het voor iemand een probleem was. Het schrijven van code gaat nou eenmaal makkelijker voor mensen met meer ervaring. Als we eerder waren begonnen was er wellicht ruimte geweest om meer van elkaar te leren.
Tijd was wel iets wat lichte stres heeft opgeleverd. Dit kwam omdat we te laat begonnen omdat we dachten dat het project pas aan het einde van de tentamen week ingeleverd moest worden. Ondanks dat hebben we er niet echt last van gehad.

Verder was een groot deel van onze tijd tijdens werkcolleges gebruikt voor discussies over "Steen", en het label wat hij heeft opgebouwd. Wij hebben dus een erg grote tijdscomplexiteit.