IDEA punten H@cherman

Link to github: https://github.com/Stof95/Hackerman.git

Infrastructuur: 5

We hebben een duidelijke structurele schikking van de files. Alles draait om ons hoofdbestand: main.py. Dit is het enige bestand dat we vanuit de Command op hoeven te roepen, vanuit daar kunnen we alle algoritmen laten draaien, en combineren.

Daarnaast verwerken we de data op een gestructureerde manier. Al data, die we terugkrijgen wordt automatisch in het daarvoor behorende mapje gestopt, met een datum en tijd stamp erop. Dit gebeurt in onze output-file, waarin op basis van het gedraaide algoritme en de huizenvariant een rankschikking wordt gemaakt. Hierdoor staat de data die we creëren, direct klaar om verder verwerkt en geanalyseerd te worden. Zo zien we heel snel welke combinatie van Algoritmen een hoge score opleveren en welke niet.

Ook is de visualisatie van onze data handig. We gebruiken expres en niet te opgemaakt 2d beeld, waarin elk soort huis een andere kleur heeft, zodat we snel patronen kunnen ontdekken in de data. Zo zien we bijvoorbeeld dat bij mappen die een hoge waarde hebben de Maison's vaak in de hoeken staan met heel veel ruimte en de House's heel dicht op elkaar. Dit viel te verwachten, aangezien het een logische gedachte is dat de huizen die het meeste waard zijn de meeste ruimte moeten krijgen. Nu is er bewijs om die gedachte te ondersteunen. We wisten echter niet hoe strak de bewering opgaat. Ofwel: wat is de balans? Wil je de House's helemaal op elkaar zonder vrij-vrije-ruimte, zodat de Maison's de meeste ruimte mogelijk krijgen. Of loont het toch meer om een paar meter van de maison's af te pakken, zodat je de kleine huizen ook allemaal een meter kunt geven. Het eerste blijkt waar te zijn. Het loont veel meer om de Maison's alle ruimte mogelijk te geven.

We hebben een file main.py die de user vraagt om hoeveel huizen hij wilt plaatsen, met welk algoritme hij dat wilt doen en eventueel hoeveel iteraties hij wilt uitvoeren. Afhankelijk van de input, roept main.py een algoritme aan uit de map met algoritmes. De algoritmes roepen op hun beurt weer functies op uit helpers.py die door meerdere algoritmes gebruikt worden. Hierin staat bijvoorbeeld een functie die de score van een lijst met alle buildings kan berekenen. Bovendien staan de objecten zoals de huizen en water opgeslagen in classes.py. Als laatst hebben we een file die zorgt voor de visualisatie van de map.

Datastructuur: **5**
Onze data (de huizen) staan in een file classes.py, waarbij een class voor eengezinswoning, een class voor de bungalow, een class voor de maison, een class voor water en een map-class waarin de waarde van de map staat, plus alle huizen. Door deze structuur kunnen we makkelijk manipulaties uitvoeren op onze data. Bovendien kunnen we met een scorefunctie in de map-class de score per huizensoort in classes van huizen aanroepen en zo de score van de hele map uitrekenen. Dit doen we door voor elk huis in een lijst met alle huizen, de afstanden tot het dichtste huis te bepalen met een functie calculate_distance.
De .gitignore zorgt ervoor dat alleen nuttige bestanden naar github worden gestuurd. Tevens is er een requirements.txt file aanwezig.

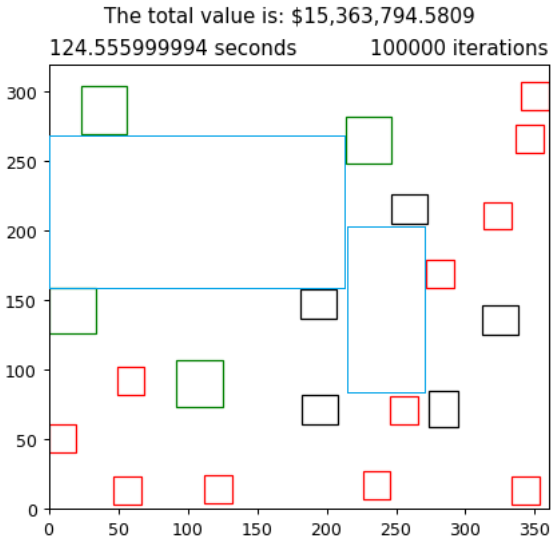
Helper file en classes file:

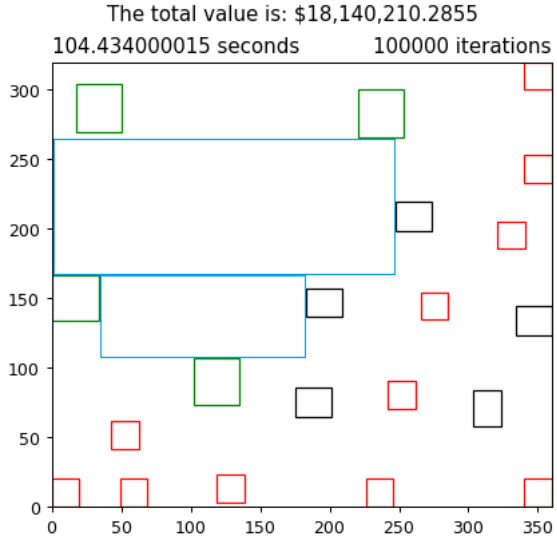
Onze data (de huizen) staan in een file classes.py, waarbij een class voor eengezinswoning, een class voor de bungalow, een class voor de maison, een class voor water en een map-class. Door deze structuur kunnen we makkelijk manipulaties uitvoeren op onze data. Bovendien kunnen we met een scorefunctie die in helpers.py staat, de score per huizensoort in

classes.py aanroepen en zo de score van de hele map uitrekenen. Dit doen we door voor elk huis in een lijst met alle huizen, de afstanden tot het dichtste huis te bepalen met een functie calculate_distance.

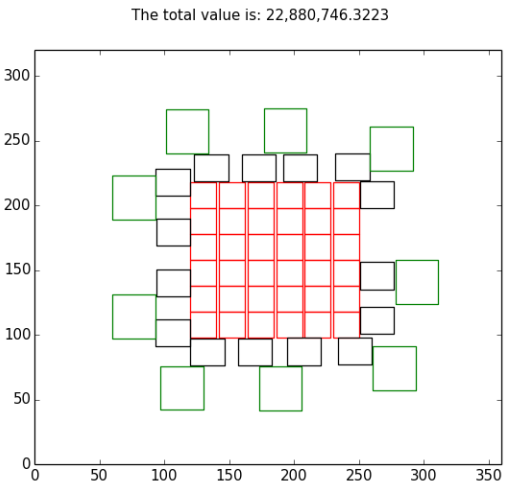
Experimentatie: 5

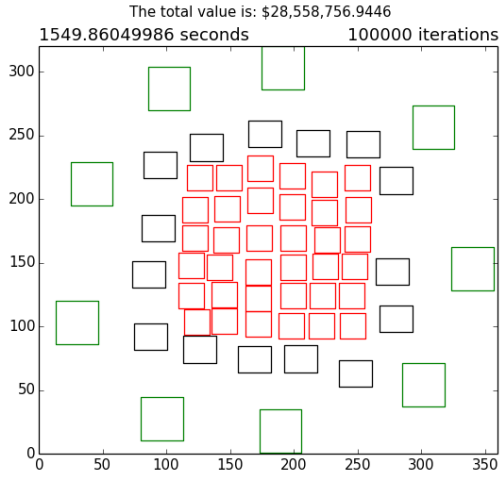
Allereerst te beginnen bij het Random genereren van mappen met huizen. Het valt te verwachten dat, aangezien er geen enkele vorm van incrementele verbetering plaatsvindt per iteratie, de verbetering van de totale score per iteratie in het begin snel zal stijgen maar na een relatief klein aantal (10.000) iteraties steeds langzaam zichzelf zal verbeteren. Om vervolgens na het generen van een goede map met het random algoritme, toch nog verbetering plaats te laten vinden, hebben we gekeken naar een iteratief algoritme. Zoals te zien is in de onderstaande tabel hebben we verschillende versies van dit algoritme uitgewerkt. De random hillclimber beweegt een random huis in een random richting waarbij de map enkel wordt opgeslagen bij een verbetering van de score. Op deze manier kun je snel veel iteraties af gaan en relatief snelle stijging verwachten van de score. Als we dit vergelijken met de systematische manier van de hilllclimber, zien we eigenlijk dat er weinig verschil tussen op te merken valt. De uiteindelijke scores liggen steeds dicht bij elkaar, dus omdat ze afhankelijk zijn van een random begin situatie valt niet te zeggen welke beter werkt. Daarbij is dit natuurlijk ook logisch omdat ze in principe hetzelfde doen maar dan in een andere volgorde; de uitkomst is altijd vrijwel hetzelfde. Wat wel duidelijk af te leiden is uit de verkregen data is dat de combinatie van het random algoritme en de hillclimber uitkomen op veel hogere eindscores. Zo eindigt bijvoorbeeld een hillclimber op een random map van 20 huizen zo'n 4 miljoen euro lager uit dan een hillclimber op een map die gegenereerd is door veel iteraties van het random algoritme. Een voorbeeld van het random algoritme, gevolgd door een hillclimber, is te zien in de onderstaande twee figuren.



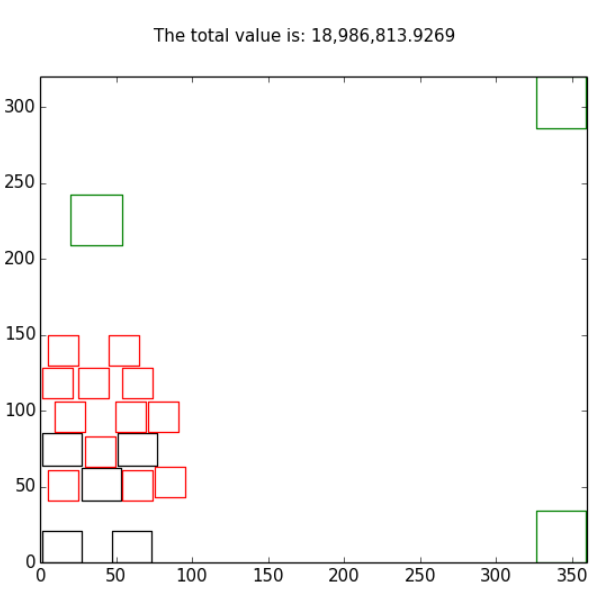


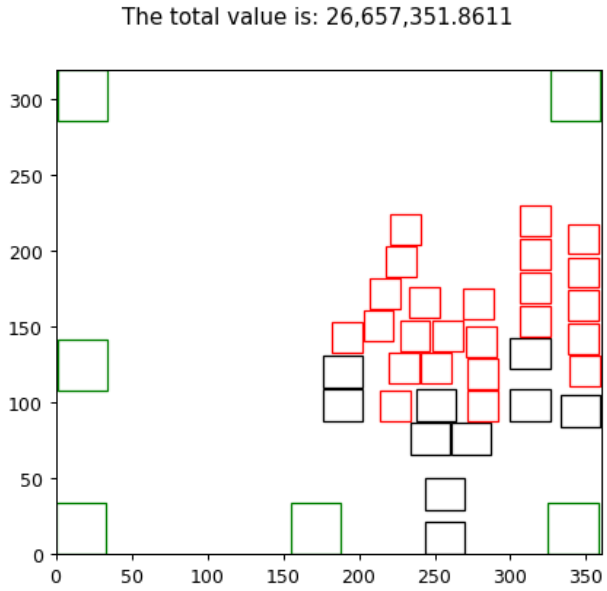
Expanding universe is momenteel nog onder constructie, maar de eerste resultaten zijn al veel belovend. Door willekeurig gekozen een aantal keer te 'expanden', haalt het algoritme al een waarde van 25 miljoen. Na hillclimbers loopt dit op tot 30 miljoen. Als de expand functie in het algoritme wordt verbeterd, verwachten we nog hogere scores. Hieronder vind je een graph van de start en eindpositie van de expanding universe heuristiek met een hillclimber algoritme.





Echter is dit meer een heuristiek dat meegegeven kan worden aan bijvoorbeeld een hillclimber, in plaats van een echt algoritme. Daarom hebben we vervolgens gekeken naar een algoritme dat huizen plaatsts op basis van waar op dat moment de map het meeste waard van wordt; het greedy algoritme. De uitkomsten waren relatief goed, maar de vorm was toch verrassend. Enige symmetrie hadden we wel verwacht bij dit algoritme. Echter wordt er, wanneer het algoritme meerdere posities vindt die dezelfde (op dat moment het hoogste) map waarde geven, gekozen voor de laatst gevonden waarde. Voor toekomstig onderzoek zou dit dus ook gerandomized kunnen worden voor een andere uitkomst. Hieronder vind je een uitkomst van 20 huizen (links) en 40 huizen (rechts) van het greedy algoritme.





Algoritme	Huizen	Random iteraties	Time random	Score random iteraties	Evt. hillclimber iteraties	Time hillclimber	Score
Random	20	100.000	124 sec.	15.363.794,-	n.v.t.	n.v.t.	15.363.794,-
Random hillclimber	20	1	0.01 sec	10.660.941,-	10.000	14 sec	14.284.720,-
Systematic hillclimber	20	1	0.01 sec	11.105.865,-	2000	126 sec	14.599.703,-
Random + Random hillclimber	20	100.000	124 sec	15.363.794,-	100.000	104 sec	18.140.210,-
Random + systematic hillclimber	20	100.000	120 sec	14.307.814,-	1.000	48 sec	18.535.696
Expanding universe	20	n.v.t.			1	0.1 sec	8.337.754,-
Expanding universe	20	n.v.t.			100	11.9 sec	13.859.720,-
Greedy	20	n.v.t.			n.v.t.	18 min	18.986.814,-
Random	40	100.000			n.v.t.	532 sec	21.133.544,-
Random + random hillclimber	40	100.000	532 sec	21.133.544,-	100.000	437 sec	23.053.714,-
Random + systematic hillclimber	40	100.000	579 sec	21.235.305,-	1000	476	24.765.414,-
Greedy	40	n.v.t.			n.v.t.	3,5 h	26.290.959,-

Random + random hillclimber	60	100.000		27.089.894,-	100.000	907 sec	28.939.904,-
Random + systematic hillclimber	60	100.000	1293 sec	27.210.681,-	1000	2040 sec	30.179.391,-
Expanding universe	60	n.v.t.			1 (systematic)	3.4 sec	25.402.433,-
Expanding universe	60	n.v.t.			100 (systematic)	336 sec	30.411.753,-
Expanding universe	60	n.v.t.			1.000 (systematic)	51 min	30.415.212,-
Expanding universe	60	n.v.t.			10.000 (random) hillclimber)	164 sec	28.407.534,-
Expanding universe	60	n.v.t.			100000 (random hillclimber)	25 min	28.558.757,-

Algorime: 5

We hebben 5 algoritmes die we soms in combinaties gebruiken: random algoritme dat een random huis op een random plek neerzet. Verder hebben we een hillclimber algoritme hillclimber_algoritme.py dat systematisch per huis alle kanten op beweegt, en daar de beste van kiest. Ook hebben we een hillclimber algoritme hillclimber_random.py die een random huis een random kant opschuift en kijkt of de waarde is verhoogd. Ook hebben we een zelf bedacht algoritme expanding_universe.py. Dit algoritme zet de eengezinswoningen in het midden, daaromheen de bungalows en daar weer omheen de maisons. Vervolgens gaat het algoritme als een soort uitdijend heelal de huizen richting de rand bewegen. Dit doen we omdat we denken dat vrije ruimte die de maisons delen meer opleveren dan vrije ruimte die kleine huisjes delen. Het laatste algoritme is greedy.py. Dit algoritme zet het meest waardevolle huis op de plek waarbij de waarde van de kaart het hoogst wordt. Het algoritme zoekt daarvoor elke positie af en onthoudt de beste plek waar nog geen huis staat.