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.

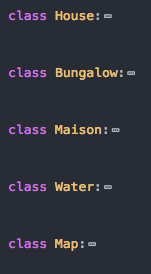
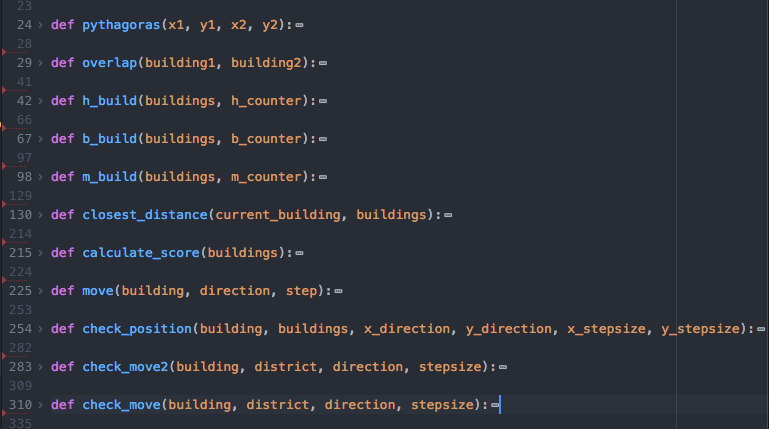
We hebben een file main.py die de user vraagt om hoeveel huizen hij wilt plaatsen, met welk algoritme hij dat wilt doen en eventueel hoeveel iteraties hij wilt uitvoeren. Afhankelijk van de input, roept main.py een algoritme aan uit de map met algoritmes. De algoritmes roepen op hun beurt weer functies 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:



**Experimentatie: 5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algoritme** | **Huizen** | **Random iteraties** | **Evt. hillclimber iteraties** | **Time** | **Score** |
| Random | 20 | 100.000 | n.v.t. | 124 sec. | 15.363.794,- |
| Random + hillclimber | 20 | 100.000 | 100.000 | 104 sec | 18.140.210,- |
| Random + systematic hillclimber | 20 | 100.000 | 1.000 | 48 | 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,- |
| Random | 40 | 100.000 | n.v.t. | 532 sec | 21.133.544,- |
| Random + hillclimber | 40 | 100.000 | 100.000 | 437 sec | 23.053.714,- |
| Random + hillclimber | 60 | 100.000 | 100.000 | 907 sec | 28.993.904,- |
| 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,- |
| Greedy | 20 | n.v.t. | n.v.t. | 18 min | 18.986.814,- |
| Greedy | 40 | n.v.t. | n.v.t. |  |  |

**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.