Notulen

11-12  
Zet in verslag dat bij a\* je een onderschatting moet maken van het aantal moves tot de oplossing. Op die manier vind je altijd de snelste oplossing.   
  
We gebruiken guided depthfirst heuristiek, omdat je dan veel minder opslaat en dus minder snel een memory error hebt.

25-11-15

Pim deptfirst fixen die wel ooit op de beste kan komen?  
Pim Kleurtjes aan visualizatie  
Daan in losse file functie maken die returned: [ [cars], [state], width, exit]  
Alex: Verder aan kutten  
Onbekend persoon: Simple A\* implementeren met als heuristiek hoeveel autos er in de lijn tot de exit in de weg staan.

16-11-2015

Pim Depth first verbeteren zodat dit de optimale oplossing geeft

Alex maakt het breath first algorithme

Daan maakt een hash functie

9-11-15  
Toestands ruimte geven?:  
Pim voegt een counter toe die incremeert elke keer als iets aan de statesArchive word toegevoegd.  
Daan car object maken/ aanpassen  
Pim in visualisatie aanpassen zodat het werkt met het nieuwe car object  
Alex: CarsList classe maken die een functie heeft die current state geeft (getCurrentState())  
Daan: Width en height variabel maken in de functie  
  
Afspraak:  
In de carslist zit de rode auto achteraan

4-11-15

Standaardvariabele:  
statesArchive: tree: het archief waar de verschillende states worden opgeslagen.  
moveOptions: Stack: de verschillende mogelijkheden.  
path: list of list: geeft de nog mogelijke kinderen van de huidige states. Elk element in de lijst stelt een level voor in de tree. Dus een lijst met een state en het aantal mogelijke kinderen.  
Solutionpaths: list of list: een lijst met alle mogelijke oplossingen. De oplossingen bestaan uit een rij aan paths.

De volgende dingen moeten gemaakt worden:

Een functie waarbij er een lijst wordt gegeven met alle mogelijke states waarbij elke auto maximaal 1 stap is verzet.

De tree moet gemaakt worden met een functie add to three and is in three.

Het hele depth-search algoritme.

Stack voor de stappen maken. Als de state al geweest is dan pop je de optie.

Creëer een array? Voor de zetten die je maakt.   
De opties slaan we op in de **archive**(van de coördinaten maken we tegels) we checken hier altijd of dit het antwoord is. Alle opties slaan we op in de **Stack**.  
We kiezen 1 optie en daar gaan we in door. De opties slaan we op in de **Stack.**De nieuwe stappen slaan we op in de stack(bovenop).

30-10-15

Taakverdeling besproken voor het weekend eerstvolgende vergadering is **03-11-2015**.

Pim: De visualisatie afronden. De output van Alex implementeren. De visualisatie updaten.

Alex: Implement can make move maken. Er voor zorgen dat er bewogen kan worden. De restricties om de functie maken. Kijken naar de mogelijke zetten.

Daan: Het depth-first algoritme implementeren in python.

Notitie: oorsprong is linker onderhoek!. De coordinaten worden bepaald door het kortste stuk naar de oorsprong

Mogelijke ideen/gedachten voor optimalisatie om later uit te zoeken:

* A; De stack anders implementeren zodat dingen aan de voorkant worden toegevoegd, testen of dat sneller is/ vragen aan maarten
* A; De allmoves functie van Daan maakt en vult een lijst met auto’s elke keer wanneer deze aangeroepen word, dit is niet nodig en kan buiten de functie denk ik
* A; Het is me nog niet duidelijk hoe het zit met de opties en wanneer die gecheckd worden of ze een oplossing zijn en of die in de statelist komen of niet.
* Je kan als auto niet overal staan op een rij, als er 3 horizontale zijn bijvoorbeeld dan kan ie niet overal komen
* Vragen aan Maarten, hoe slaat python een int op? Maakt ie m meteen 64 bit?
* D; Unblock me algorithm solving, kunnen we naar kijken
* A; Ik heb de lengte van de solutions geprint om de 30 van de 3100 oplossingen en de korste is 432 ofzo en dan lopen ze op tot 2000 en dan daalt het weer naar 630 ofzo. Maar de eerst toegevoegde oplossing is het kortst en daarna worden ze telkens groter en dan weer korter, waar komt dit patroon vandaag? Is dit logisch?
* A; Profiler deed er 16 seconden over met parentOfparent verhaal. En wanneer de childcount methode werd gedaan terwijl voor elke optie alsnog de parent werd toegevoegd aan de tree was het totaal: 113 seconden ofzo dacht ik, maar er is ook een zootje deepcopy wat erbij komt
* A; Profiler deed er 4.748 seconden over met bounded DFS. Dit was wanneer de childcount methode werd gedaan terwijl voor elke optie alsnog de parent werd toegevoegd aan de tree. Zo werd een beste oplossing gevonden van 423.
* A; Door 1 teken te veranderen, if len(PATH\_TRACKER.path) < MaxDepth naar <= MaxDepth zorgde er ineens voor dat de 423 oplossing niet meer werd gevonden, hoe kan dat?