Eray Sarikaya, 1790036
Kenan de Vries, 1811470
Matthew Jim, 1812418
V2B

Programming Inleveropdracht 2: Agents & Simulation Tools (groepsopdracht)

Inleiding

Voor het programming gedeelte van het vak Simulation gingen wij als groep aan de slag met het experimenteren en onderzoeken van tools die helpen bij simulaties ontwikkelen. Het is hierbij van belang dat we kijken naar de voor en nadelen van onze tools en hoe we dit toepassen voor ABM (agent based modeling). Wij hebben Unity, NetLogo en Mesa gebruikt en onderzocht. Want zoals een wijs docent ooit zei "A fool with the tools is still a fool."

Volg de tutorial en omschrijf daarna in één paragraaf wat deze tool anders maakt dan andere programmeertalen, wat zijn de voor- en nadelen?

Mesa voordelen:

- Mesa maakt makkelijk en snel simpele visualisaties op basis van een browser GUI.

- Relatief snel vergeleken met simulaties in andere talen zoals Unity.

- Omdat het een framework van Python is kan het gebruik van andere libs als matlab en scikit-learn.

- Makelijk om mee te beginnen als beginner.

Mesa nadelen:

- Hoewel de visualisatie makkelijk en snel is, is deze minder indrukwekkend dan een visualisatie in bijvoorbeeld Unity. Er zijn geen gevisualiseerde 3D grids mogelijk of modellen.

- Singlecore processing i.p.v. multithreading wat tegenwoordig de norm is. Dit houd in dat er maar 1 process tegelijk kan worden uitgevoerd. De performance gaat naar beneden omdat andere processen moeten wachten tot de eerste klaar is, terwijl dit bij veel andere talen niet het geval is.

- Mesa is nog een relatief nieuw (2013) en vertrouwd op de community voor verbetering. ter referentie, Unity kwam uit in 2005 en is nog steeds aanzienlijk populair in het bedrijfsleven wat zorgt voor veel ondersteuning de komende jaren.

Unity voordelen:

- Enorm populair dus veel communitys waar er om hulp gevraagd kan worden.

- Kan in 2d en 3d geprogrameerd worden.

- Heeft een store waar verschillende modellen en andere assets geimporteerd van kunnen worden.

- Makkelijk om in te stappen als beginner.

- heeft play mode, waardoor je gelijk in realtime kan testen en zien wat er gebeurt en eventuele bugs er uit kan halen.

Unity nadelen:

- Door de architectuur in Unity is het moeilijk om SOLID principes toe te passen (dit hoort bij OOP programmeren). Omdat het chaotisch kan worden bij lange programmas wordt dus ook het vinden van bugs en errors moeilijk.

- Is niet Thread-safe. Dit beschermt tegen slecht gebruik van multi threading, maar zorgt ervoor dat het optimaliseren van applicaties en simulatie smoeilijker wordt.

- Gebruikt veel processing power en heeft dus ook meer behoefte aan optimalisatie.

De tutorial van Unity is helaas niet afgemaakt omdat er problemen waren bij het laden en compilen van de script.

NetLogo voordelen:

- Lijkt op 'natuurlijke taal' waardoor je code eigelijk al lijkt op psuedo code.

- Makkelijk voor beginners maar moeilijk om de geavanceerde functies te implementeren. Ook wel 'low skill floor, high skill ceiling' genoemd.

- Goeie en elegante GUI, wat erg fijn is om je simulatie overzichtelijk te maken.

- Mooie visualisatie met de optie voor 3d visualisatie.

- Heel erg veel tutorials, voorbeelden en documentatie.

NetLogo nadelen:

- NIET OOP, maar prodecueel! Hierdoor is het snel onoverzichtelijk en worden grote modellen en projecten chaotisch en tijdskostig.

- Is er al sinds 2002 wat het wat verouderd maakt. Een aantal leerlingen in de klas was toen nog niet eens geboren.

- Je moet iets met Turtles hebben.

We definieren een staat en drie functies waarmee we een stateful agent abstract omschrijven:

1. Een initiele staat $i_0 \in I$, waarbij ***I*** alle mogelijk interne staten van de agent zijn
2. Een functie "See" of "Perceive", die een mapping maakt van elke staat in de omgeving tot een staat die de perceptie van de agent van de omgeving aangeeft. Dus: *See:* $S \longrightarrow P$, waar ***S*** de staat of serie van staten van de omgeving is en ***P*** de perceptie van die omgeving
3. Een functie "Act" die een kijkt naar de staat van de agent en een toepasselijke actie kiest. Dus *Act:* $I \longrightarrow A$, waar ***I*** een interne staat is en ***A*** een actie.
4. Een functie "Update" die een staat ***I*** neemt (soms ***D*** genoemd) en perceptie ***P***, en een nieuwe staat ***I*** oplevert, dus *Update:* $I \times P \longrightarrow I$
   Dit is een korte samenvatting van wat er staat in: "An introduction to Multi-Agent Systems" chapter 1.1 up to 1.4. Die

Beschrijf in eigen worden wat elk van de 4 concepten in het algemeen beschrijft EN wat het in jouw specifieke simulatie betekent.

1. Het begin van een stateful agent abstract begint bij een initiele interne staat i0. Dit is waar een agent start op tijdstap 0. Die state gaat in de functie See. De informatie wat in die state zit is wat die agent allemaal weet, zoals zijn positie. In het mesa voorbeeld is de intiele staat waar een agent start.

2. De functie See neemt een S en returnt een P, waarvoor S een state of environment is en P een perception van de omgeving is. De state of environment zijn alle agents in het model in verschillende groepen tot waar die bij horen. De perception is wat de agent allemaal ziet en kan doen, het kan voorkomen dat je de agents weinig zicht geeft waardoor die misschien alleen maar een stukje voor zich kan zien. Dus dan word de perception van die agent minder of zelfs niks. De perception zijn de coördinaten van de verschillende agents die een andere agent ziet. In het mesa voorbeeld ziet elke agent elkaar.

3. De functie act neemt een I en return een A, waarvoor I een interne state is en A een actie. Wanneer hij een I ontvangt gaat die een actie uitvoeren, dan gaat de agent opnieuw in de cyclus in. Waar die dan weer opnieuw de functie See dan update die state met update en daarmee weer een nieuwe actie gaat uitvoeren. In het mesa voorbeeld is de functie act dat ze aan elkaar geld geven als ze in de radius zitten.

4. De functie update neemt een I en P en return een nieuwe I, waarvoor I een set is van alle interne states van de agent en P een perception. De interne state van de agent word geupdate nadat See word aangeroepen. Dus het neemt de interne state van tijdstip 0 en functie See(s). Hieruit komt dan een nieuwe I uit, deze I kan je dan weer in de functie act doen. In het mesa voorbeeld is update steeds dat het opnieuw aangeroepen word met de informatie.

Beschrijf je omgeving op basis van de dichotomiën die hier op pagina 6 beschreven staan, en licht toe (dus niet alleen termen opsommen):

Het model van ons groepje bestaat uit een simulatie van een economie waarin agents geld eenheden ruilen en opsparen. Een agent begint met 1 geld eenheid en elke stap verplaats de agent zich in de hoop bij een of meer andere te komen. Indien dit gebeurt gaat de agent altijd zijn geld aan een andere agent proberen te geven, dit kan ook mislukken als hij momenteel niks heeft. Een agent kan ook geld ontvangen van 2 andere agents die bij die agent in de buurt staat, wat zorgt dat het een nettowinst heeft van 1 eenheid. Een omgeving waarin we informatie hebben over waar de agent zich momenteel bevind in een vaste ruimte, hoe financieel gezet deze is, en waar de agent heen kan gaan noemen we accesible.

In de simulatie worden de agenten random verplaatst, wat voor verschillende eindresultaten zorgt. Het maakt niet uit of de agenten op dezelfde plaats starten en een vaste reeks acties onderneemt, want in het verplaatsen van de agents word random gebruikt. Daarnaast word er ook nog random gebruikt in het kiezen van de neighbor, als er meer dan 1 agent in zijn radius staat. Gezien zowel de loop van de simulatie als de eindresultaten erg varieren is onze simulatie Non-Deterministic.

Een agent in onze omgeving zet zonder uitzondering altijd 1 willekeurige stap en doet altijd een poging om 1 van zijn geld eenheid te geven aan een andere agent. Dit gedrag veranderd om geen enkele omstandigheden, dus er hoeft alleen gekeken te worden of het mogelijk is voor een agent om zijn actie uitevoeren (heeft hij genoeg geld, staat er iemand in de buurt). Dit valt onder de catergory Episodic.

Ons rijkdomverdelings model heeft een vrij versimpelde wereld waarin alleen agents lopen en geld uitwisselen. Verder is het geen dynamische wereld waarin een agent ineens de loterij wint of een agent niet beweegt omdat hij in de file loopt(?). Omdat dus alleen onze agents invloed hebben op de omgeving is het een static dichotomie.

Als laatste is het van belang om te definieren of onze omgeving discreet of continue is. Onze wereld bestaat uit een grid van 20x20 waar de agents onder geen enkele omstandigheden uitkunnen. Het maximaal aantal agents dat geplaats kan worden aan het begin is 200, met een miminum van 2. Deze agents kunnen tegelijkertijd op een vakje staan en kunnen altijd maar aan 1 agent geld geven. Een agent kan dus maar op 400 vakjes staan en 2 acties (geld geven als er iemand is en anderzijds niks doen). Er wordt een gini van de rijkdomsverdeling bijgehouden (tot 3 decimalen) en de hoeveelheid geld van een agent is onbeperkt. We kunnen dus vastellen dat het aantal acties beperkt zijn, maar de resultaten onneidig doorlopen en veranderen. Dit valt onder onder de definitie discreet.

Bedenk een voorbeeld waarbij minimaal 3 dichotomies precies tegenovergesteld zijn en beschrijf of het veranderen van je omgeving op deze manier wel of niet iets zou toevoegen aan je simulatie.

Voor de veranderingen van onze omgeving dachten wij eraan om de simulatie zo aan te passen dat er een bepaalde mate van inkomensgelijkheid komt tussen de agents. We zouden van episodic naar non episodic gaan door de agents met veel geld naar een specifieke neighbor met weinig geld te laten lopen in de volgende de stap, of agents die pas hebben geruild niet gelijk weer te laten ruilen zodat agent a niet al zijn geld aan agent b geeft. We willen net als in het echt ook iets in de richting van een belasting regel toepassen, zoals iedere agent met 4 of meer geld moet 2 muntjes ruilen in plaats van de standaard 1. Hierdoor zou er meer gelijkheid moeten komen in de verdeling van het geld.Dit maakt dat we van static naar dynamisch gaan. Als laatste willen we van deterministisch naar non deterministisch gaan door alle willekeurigheid wegtehalen. Door bijvoorbeeld vaste loop patronen en een vast aantal agents zullen de resultaten en gebeurtenissen altijd hetzlefde blijven. Agent A die maar 1 muntje heeft zal altijd op beurt 10 ruilen met agent B die 4 muntjes heeft. Echter kunnen we specifiek kijken welk effect onze maatregelen hebben in de simulatie en de einresultaten als we besluiten deze te veranderen. Bij een non deterministic simulatie kunnen wij dit niet, omdat de gebeurtenissen altijd anders zijn en de maatregelen de resultaten altijd anders beinvloeden.