Simulation Tooling Opdracht 3 – Agent Based Modelling & Simulations

Door Gerrit van de Bunt, Projectgroep 5

Vraag 1

Volg de Tutorial en omschrijf daarna in één paragraaf wat deze tool anders maakt dan andere programmeertalen, wat zijn de voor- en nadelen? Leg ook uit waarom je programma wel of niet agent-based is.

Als programmeertaal heb mesa gebruikt; hierin heb ik een agent based model & simulation geschreven waarin agents in een omgeving (willekeurig) bewegen en telkens (als er meer dan 2 aangrenzende agents zijn) geld weggeven. Het programma is hier dan agent-based; hoewel er geen states in voorkomen doorlopen agents wel telkens een perceptie (kijken naar omliggende cells waar ze naartoe kunnen en agents waar ze geld aan kunnen geven) en een actie fase (het lopen en het geven van geld), en dit past de omgeving dan weer aan. Mesa is net weer iets anders dan tools zoals bijv. NetLogo of Unity; Mesa is immers geschreven in Python en kan dus gebruik maken van allerlei modules/libraries die op het internet beschikbaar zijn. Bovendien is het makkelijker om simulaties aan te maken omdat Python procedureel/object-georiënteerd combineert, wat bijvoorbeeld net iets anders is dan met NetLogo en met Unity (bij het laatste wordt gebruik gemaakt worden van C#, wat volledig OO is).

Een voordeel aan mesa en wat het ook anders maakt dan andere tools is de integratie met python; niet zo bijzonder op zichzelf, maar - wel heel erg nuttig - gegeven dat python een grote waslijst heeft aan modules waar je mee kunt werken in tegenstelling tot NetLogo & Unity, hoewel hier wel gewoon code voor geschreven kan worden.

Bovendien – zoals eerder vermeld – is mesa een package dat hoort bij Python. Python is makkelijk om mee te beginnen (wordt overigens veel gebruikt voor AI en dus ook een beetje voor ABMS) en kan procedureel & object-georiënteerd geprogrammeerd worden, hoewel er bij Mesa wel gebruik gemaakt moet worden van object-georiënteerd programmeren. Tenslotte hoort hier ook nog bij dat in Python er heel veel aparte modules bestaan die je kan gebruiken met Mesa om tot een goede data visualisatie te komen; in tegenstelling tot NetLogo zijn de visualisaties daar een beetje rudimentair, maar met bijvoorbeeld holoviews kan de visualisatie ook nog interactief zijn.

Een nadeel van mesa is echter de lage schaalbaarheid; dat zie je terug als je een simulatie gaat runnen met veel agents (in mijn geval varieerde dat tussen de 10 en 490 agents, na elke simulatie werd het aantal agents verhoogd), wat Mesa ongeschikt maakt voor simulaties waarbij veel agents aan te pas komen, bijvoorbeeld – een heel extreem geval – het simuleren van de hele wereld; daar komen natuurlijk heel veel agents en andere externe factoren bij kijken die allemaal meegenomen moeten worden, en dat is hier niet zo makkelijk om te doen hier.

Vraag 2

Beschrijf in je eigen woorden wat de initiële staat was, de “See” of “Perceive”, de “Act” en de “Update” functies waren binnen jouw tutorial.

In de uitgewerkte tutorial was er sprake van Purely Reactive Agents, dat houdt in dat de agents alleen maar werken met een See en Act functie; hier waren dus geen states en er was ook geen Update functie. De See functie zou hier inhouden dat de agent zou checken welke agents er naast hem zaten, en daarvan zou (willekeurig) een agent gekozen worden als doel voor de Act functie; de Act functie gaf altijd een (willekeurige) hoeveelheid geld aan de doelwit agent.

Met de introductie van mijn eigen wijzingen is er nu sprake van State-based agents; er is een initiele staat (twee states; 1 = weggeven van geld, 2 = stelen van geld, waar de startstate 1 is), en er is nu een Update functie die de state aanpast (vooral gebaseerd op hoeveel geld de agent heeft, dit is leidend in welke state hij kiest; zijn doel is immers om geld uit te geven en om ook een ‘gezonde’ hoeveelheid geld (abstract: 10) te houden). De See functie is hetzelfde als voorheen, en de Act functie gaf nu geld weg of steelt geld van andere agents (natuurlijk gebaseerd op de state en de perceptie). Daarom is er nu ook sprake van een state-based agent.

Vraag 3

Beschrijf je opgeving op basis van de dichtonomies die horen bij ABSM, en licht toe.

1. Accesssible vs **inaccessible**

   In de omgeving zijn niet alle gegevens beschikbaar; de agent weet wel hoeveel geld hij heeft en in welke state hij zit, maar hij moet wel continu kijken welke agents er naast hem (kunnen) staan. In een accessible omgeving zou de agent continu met de andere agents kunnen interacteren.

2. Deterministic vs **non-Deterministic** (Stochastic)

   De omgeving is non-deterministisch; er zit een element van willekeurigheid in; agents verplaatsen zich altijd naar een willekeurige locatie, dus met elke run zullen de agents zich op een andere plek bevinden. Bovendien geven ze ook telkens aan een willekeurige buur geld of stelen ze het. Bij een deterministische omgeving zou er logica hierachter moeten zitten, bijvoorbeeld; agents verplaatsen zich naar een agent met het meeste geld, stelen hier geld van (als ze dat willen, denk aan de states), en geven bijvoorbeeld geld aan de agent met het minste geld om zo het geld goed te verdelen (in een maatschappij bijvoorbeeld).

3. **Episodic** vs non-episodic

   Bij een episodiche omgeving hoeven agents geen rekening te houden met de huidige en toekomstige tijdsstappen. Hier wordt daar überhaupt geen rekening mee gehouden. Bij een non-episodische omgeving zouden agents bijvoorbeeld op kunnen houden met geld geven als ze dicht bij de grens komen waar ze wel zouden moeten gaan stelen (onder de 10 geld).

4. **Static** vs dynamic

   De acties van de agent zijn leidend bij het veranderen van de omgeving; andere agents kunnen immers niet zomaar geld ergens vinden op het bord (10x10 cellen). Bij een dynamische omgeving zou er bijvoorbeeld wel willekeurig geld kunnen verschijnen op het bord (denk bijvoorbeeld aan baan-aanbiedingen, maar denk bijvoorbeeld ook aan een modelvoorbeeld uit netlogo; de modellering tussen predatie van schaap op wolf en gras op schaap).

5. Discrete vs **continuous**

   Feitelijk zou de simulatie continu kunnen blijven doordraaien; agents geven immers geld weg als ze dat hebben en stelen geld als ze niet genoeg geld hebben. Er gaat geen geld verloren van het totaal wat er in het bezit is (100 * agents, elke agent begint met 100). Bij een discrete omgeving zou er bijvoorbeeld belasting geheft kunnen worden, waardoor eventueel het totaal van wat er in het bezit is wel bij 0 komt (of enigszins er dichtbij), waardoor de simulatie wel zou aflopen.

Vraag 4

Bedenk een voorbeeld waarbij minimaal 3 dichtonomies precies tegenovergesteld zijn en beargumenteer waarom dit wel of niet van belang is om je simulatie nuttig te maken

Een voorbeeld zou de aandelenmarkt zijn. Hier zouden de agents aandeelhouders kunnen zijn, en omdat de aandeelmarkt geen fysieke locatie kent (je koopt online aandelen) is de simulatie **accessible**; agents kunnen niet alleen met de bedrijven maar ook met elkaar handelen (zie het bijvoorbeeld als een forum of zoiets dergelijks). Bovendien is een aandelenmarkt **dynamisch**; bedrijven kunnen groeien en krimpen, wat de aandelenprijs beïnvloedt. Natuurlijk hebben agents hier invloed op, maar als een bedrijf het bijvoorbeeld slecht doet zijn agents geneigd om hun aandelen te verkopen, omdat de winst die ze krijgen met de aandelen terugloopt. Tenslotte is de simulatie ook **non-episodic**, agents die volgen het nieuws en moeten telkens rekening houden met welke richting het bedrijf opgaat waar ze hun aandelen hebben aangeschaft; als ze kunnen inzien dat het bedrijf op de rand van de afgrond staat, willen ze natuurlijk zo snel mogelijk hun aandelen verkopen.

Het zou wel interessant zijn om deze veranderingen te implementeren; met deze simulatie zou het gedrag van aandeelhouders voorspeld kunnen worden, dit kan door op basis van gescripte events (bijvoorbeeld bedrijf a lijdt over 100 dagen aan een catastrofe) de simulatie te runnen.