AGENTS EN AGENT-TOOLS

King of the Mesa

26 NOVEMBER 2020

KEVIN WANG (1763456)

Omschrijf 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.

Mesa is een object georiënteerde Python framework voor het modelleren van agent-based simulaties. Mesa is anders dan Unity, omdat Mesa een Python-module is, i.p.v. een engine en wordt dus ook voor andere use-cases gebruikt. NetLogo is ook anders dan Mesa, omdat NetLogo zijn eigen programmeertaal is, speciaal gemaakt om agent-based simulations te maken.

Voordelen:

- Mesa is een Python module. Dat betekent dat je deze meer functionaliteiten kan geven door het te combineren door de talloze andere Python modules zoals Numpy, Pandas, etc.
- Mesa kan de modellen op een HTML-webpagina laten zien en werkt ook met JavaScript. Dit kan dus makkelijk geïntegreerd worden op een al bestaande website.
- Mesa is makkelijk op te pakken, omdat het in bekende programmeertalen geschreven is (OO-Python, JavaScript).
- Doordat Mesa in bekende programmeertalen geschreven is, is er veel te vinden qua documentatie en StackOverflow. NetLogo is bijvoorbeeld niet zo bekend, en je hebt dus weinig community ondersteuning.

Nadelen:

- Mesa gebruikt een grid om een omgeving en de beweging van agents daarin weer te geven, Unity geeft bijvoorbeeld de mogelijkheid om in een oneindig grote 2D/3D space te werken. Je kan dus beter Unity gebruiken voor bepaalde use-cases waarbij je gedrag moet simuleren.

Mesa is agent-based. Mesa is object-georiënteerd, en maakt instanties van agents aan, elk met een bepaald gedrag en set aan regels waar ze zich aan moeten houden.

Eigen aanpassingen:

- Agent categorieën; Agents veranderen van kleur als ze een bepaald aantal geld hebben. Dit heb ik gedaan door extra elif-statements toe te voegen die checkt op agent.wealth. Afhankelijk van de wealth, verander ik de Color, Layer en r.

  Zwart < €1

  Grijs = €1 – €2

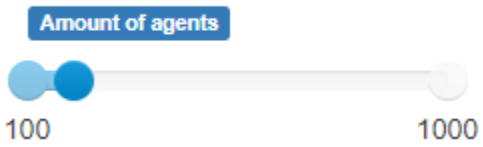
  Groen = €3 - €5

  Blauw > €5

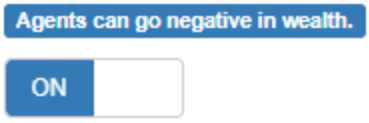
```
def agent_portrayal(agent):
    portrayal = {"Shape": "circle",
                 "Filled": "true",
                 "r": 0.2}

    if agent.wealth == 0:
        portrayal["Color"] = "#000000"
        portrayal["Layer"] = 3
    elif 0 < agent.wealth < 3:
        portrayal["Color"] = "grey"
        portrayal["Layer"] = 2
        portrayal["r"] = 0.5
    elif 3 <= agent.wealth < 6:
        portrayal["Color"] = "green"
        portrayal["Layer"] = 1
        portrayal["r"] = 0.8
    else:
        portrayal["Color"] = "blue"
        portrayal["Layer"] = 0
        portrayal["r"] = 1
    return portrayal
```

- UserParam toegevoegd; geeft de mogelijkheid aan de gebruiker om aantal agents aan te passen (100-1000). Mesa heeft hier een handige module voor (mesa.visualization.UserParam.UserSettableParameter). Deze module heeft sliders, maar ook input boxes, boolean options, en static tekst.



- UserParam toegevoegd; geeft de mogelijkheid aan de gebruiker om aan te geven of agents in de schulden mogen lopen als ze geld weggeven. Dit is weer een klasse uit mesa.visualization.UserParam.UserSettableParameter. Dit heb ik gemaakt door een parameter toe te voegen aan MoneyModel (goNegative). MoneyModel geeft deze parameter weer door aan MoneyAgent, waar een if-statemen in staat die kijkt of de dit True of False is. Afhankelijk daarvan checkt hij of een agent geld heeft of niet.



Als de UserParameters veranderd worden, moet de webpagina opnieuw geladen worden voordat het effect heeft.

Beschrijf nu in je eigen woorden wat deze functies zijn in de context van de simulatie uit je tutorial.

1. Agents beginnen met een willekeurig bedrag (0 – 2), en hebben een willekeurige positie in de wereld.
2. Agents hebben een variabele cellmates die de agents op dezelfde cel weergeven.
3. Als er agenten op dezelfde cel zitten, geeft de agent geld mits deze geld heeft om weg te geven, en tenzij de gebruiker anders heeft aangegeven.
4. Als de 'geef-geld' fase over is, neemt de agent weer een stap, en dit proces herhaalt zich.

Beschrijf je omgeving op basis van de dichotomies en licht toe.

Accessible; alle agents kunnen vrij bewegen naar elk vakje op de grid, ook als er al andere agents op dat vakje staan.

Non-deterministic; Agents beginnen met een willekeurige startpositie en een willekeurig aantal geld (0 – 2). Dit maakt deze simulatie non-deterministisch, omdat de uitkomst nooit hetzelfde uitkomt.

Non-Episodic; De vorige acties van agents hebben effect op de volgende. Als de agent al zijn geld heeft weggegeven, kan deze niet nog meer geld weggeven, mits dit anders is aangegeven met de UserParam

Static; De omgeving veranderd niet, ook niet door het gedrag van andere agents. Elke agent kan op elk vakje staan onder welke conditie dan ook.

Discrete; De omgeving bestaat uit een 30x30 grid, waarin de agents vrij kunnen bewegen.

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

Non-accessible:
De simulatie zo maken dat agents niet op dezelfde cel kunnen staan. Dit zorgt ervoor dat een agent niet altijd de acties uit kan voeren die hij wil. Dit is niet nuttig in onze simulatie, omdat agents op dezelfde cel moeten staan om geld te geven. Als er geïmplementeerd kan worden dat ze een kijkradius van 1 cel hebben, dan kan dit wel.

Deterministic:
Als alle agenten op dezefde plek op de grid zouden beginnen, en altijd met €1 beginnen, zou de simulatie deterministisch zijn, omdat er dan geen kans-factoren meer zijn. Dit kan nuttig zijn voor de simulatie als je bepaalde regels voor je simulatie wil aanpassen, zodat je makkelijker het effect kan zien van de verandering in je agent rules.

Dynamic:
Vakjes in de grid kan je willekeurig laten 'sluiten', zodat agents niet in de aangegeven vakjes mogen komen. Zo kan je bijvoorbeeld een city-grid simuleren.