1. 1. Omschrijf wat de tool onderscheidt van andere programmeertalen.
Één van de dingen wat mesa onderscheidt van andere programmeertalen is dat het een modulair framework is, verschillende gedeeltes functioneren apart van elkaar maar werken ook samen in drie verschillende categorieën modules; modelling, analysis en visualisation. Je gebruikt in mesa geen aparte programmeertaal maar je maakt gebruik van Python en bijbehorende libraries.

In het modelling gedeelte definieer je je classes (je kunt modules als mesa.Model en mesa.Agent importeren). In het analysis gedeelte verzamel en analyseer je de data. In het visualisatie gedeelte run je je simulatie en kun je het gedrag observeren, ook kun je door middel van libraries als matplotlib en numpy grafieken laten tekenen van de verandering van je data. Het splitsen van modules lijkt me een voordeel van mesa aangezien je alles overzichtelijk houdt, ook als je veel grotere, complexe simulaties wilt gaan bouwen. Ook kun je makkelijk aanpassingen maken in de verschillende modules en zo je programma's tweaken.

1. 2. *Leg ook uit waarom je programma wel of niet agent-based is.*
Onze simple economy simulatie is agent-based. Je programmeert vanuit het gedrag en de behoeftes van de agents. Je leert ook hoe de simulatie zich gedraagt door de agents te observeren. De afzonderlijke entiteiten worden bestudeerd en op basis hiervan trek je je conclusies. Je kunt het grote geheel bestuderen door te bekijken hoe individuele entiteiten zich gedragen, of erachter komen hoe individuele entiteiten zich gedragen door het groot geheel te bekijken.

2. Beschrijf de kernfunctionaliteit van de simulatie
De simulatie geeft een bepaald aantal agents weer, zij hebben een bepaalde integer Wealth (we noemen het vanaf nu even money). Het environment is door middel van een grid verdeeld in vakjes; wanneer er meerdere agents samen in één vakje komen zijn het buren. Wanneer agentA in bezit is van money, geeft agentA 1 money weg aan zijn buur-agentB. De agents bewegen zich over het speelveld en blijven money weggeven en aannemen. De simulatie moet een versimpelde versie van onze economie voorstellen, in de zin dat geld steeds van de ene naar de andere entiteit wordt doorgegeven.

3. 1 Initiële staat
Agents kunnen verschillende interne staten hebben, alle mogelijke staten worden gerepresenteerd in set I. De initiële staat is de staat waarin een agent zich verkeerd wanneer de simulatie begint met runnen. De wealth van een agent is in de code gedefineerd als wealth = 1. Ook krijgt de agent een positie mee. Zijn initiële staat is dus bij wealth = 1, op een willekeurige positie(x,y) met een x aantal buuragents.

3. 2 See functie S —> P
De see functie observeert zijn omgeving. De see functie returnt een percept, op basis van wat er om de agent heen gebeurt.

Wanneer agentA een buur-agentB tegenkomt in hetzelfde vakje van het grid, wordt dit gereturned als een percept.

3. 3 Update functie I x P —> I
De functie update dient voor het aanpassen van de interne staat van de agent. De see functie geeft een percept door waardoor mogelijk de staat moet worden aangepast. Zo kan het bijvoorbeeld zijn dat de agent van i0 (wealth =1, positie(x,y), buuragents = 0) naar een andere staat i1 gaat (wealth = 1, positie(x1,y1), buuragents = 1). Dit doet de update functie en returnt de nieuwe interne staat.

3. 4 Act functie I —> A
De act functie gaat een bepaalde actie uitvoeren op basis van de interne staat. Stel dat we kijken naar de interne staat i1 uit de vorige paragraaf, zou de actie *GiveMoney* moeten worden uitgevoerd, omdat de agent in bezit is van money en een buuragent heeft. Als we kijken naar interne staat i0, moet de actie *move* uitgevoerd worden om de agent te bewegen binnen het environment.

4. Beschrijf de simulatie op basis van de 5 onderstaande dichotomies.

• *Accessible vs Inaccessible*

Een accessible environment is vaak veel simpeler dan een inaccessible environment. Als een agent complete en accurate kennis heeft over zijn environment is het een accessible environment. Deze simulatie is accessible omdat je alles ten alle tijden kunt observeren, je weet ten alle tijden de posities en staten van alle agents en het environment. Beslissingen die gemaakt worden worden gedaan op basis van informatie die beschikbaar is.

• *Deterministic vs non-deterministic*

Een deterministic environment wil zeggen dat, wat dan ook, een bepaalde input leidt tot een bepaalde uitkomst. Er moet geen twijfel zijn over het resultaat wat de simulatie teruggeeft. Je kunt de simulatie een aantal keer runnen, maar je weet nooit de precieze uitkomst. Omdat er een element is van willekeurigheid (de randomchoice van de agents) kan de uitkomst steeds verschillend zijn en kun je deze dus niet voorspellen. Onze simulatie is dus non-deterministic.

• *Episodic vs non-episodic*

Iets is episodic wanneer de actie alleen afhangt van de huidige episode. Iets is non-episodic (of sequential) wanneer de beslissingen afhangen van eerder gemaakte beslissingen, dit is is het geval binnen onze simulatie. De acties van de agent hangen af van eerder gemaakte acties; zo wordt er bijvoorbeeld een andere beslissing gemaakt wanneer de agent in de vorige stap zijn laatste money weg heeft gegeven dan wanneer hij na de vorige stap nog money over heeft.

• *Static vs Dynamic*

Het environment in deze situatie is dynamic, dit komt doordat alle agents zich op een verschillende manier binnen het veld bewegen. Deze bewegingen hebben invloed op het environment en zo verandert het environment telkens. Mocht er maar één agent zijn die zich beweegt binnen het environment, is het een static environment omdat de omgeving zelf niet verandert.

• *Discrete vs Continuous*

Iets is discreet als er een eindig, vastgesteld aantal acties zijn. Ons environment is continuous omdat de agents zich oneindig lang over het veld kunnen blijven bewegen. Het krijgen en geven van money zou in principe oneindig door kunnen gaan.

5. Bedenk een voorbeeld van een simulatie met minstens 3 dichtomonies die het tegenovergestelde zijn van onze huidige simulatie.

Ik heb bij de examples van mesa gekeken en het hex_snowflake example gevonden. Bij dit voorbeeld heb je een aantal cellen die alive of dead kunnen zijn, een cel gaat van dead naar alive wanneer deze cell precies 1 alive cell als buurman heeft (zie vb1.). Alive cellen blijven voor altijd alive.

Het hex_snowflake voorbeeld is net als onze simulatie ook accessible, maar het snowflake example is deterministic. Dit is om dat er geen twijfel is over tot welke uitkomst de simulatie zal leiden, de uitkomst is voorspelbaar en er valt verder niks aan te passen dus is de simulatie deterministic.

Ik denk wel dat de snowflake simulatie ook sequential (non-episodic) is, net zoals onze simulatie, dit omdat de cellen een staat aannemen op basis van de eerder aangenomen staten van eerdere cellen. Het environment is static omdat er op geen manier iets veranderd aan de omgeving, er zijn geen andere agents die zich over het veld bewegen of het veld kunnen aanpassen. Ook is het snowflake example discreet omdat er een eindig aantal stappen zijn die de simulatie kan doorlopen.

vb1*: Alle dead cellen die 1 alive cell als buurman hebben, worden nu ook alive.*