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 (numpy, matplotlib etc.).

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. Je kunt je simulatie op verschillende manieren visualiseren, je kunt bijvoorbeeld je simulatie gewoon runnen binnen een browser om te zien hoe het zich gedraagt, maar je kunt ook je simulatie runnen en vervolgens grafieken laten ontwikkelen om de uitkomst van je simulatie te tonen.

1. 2. *Leg ook uit waarom je programma wel of niet agent-based is.*
Het programmeren in mesa 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. Binnen mesa kun je 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 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 is in de code gedefineerd als wealth = 1. Dit is dus de intiële staat. We zouden dit getal kunnen aanpassen, met als voorwaarde dat wealth altijd > 0.

Een andere mogelijke staat i1 is dat de agent een wealth van = 0 heeft, deze kan niet negatief worden omdat een agent niet iets kan weggeven wat het niet bezit. Ook hebben de agents een bepaalde positie, dit geeft weer waar de agents zich binnen het environment bevinden en of ze wel of geen buur agents hebben.

3. 2 See functie
De see functie ontvangt informatie over de buur-agents van een bepaalde agent A. Wanneer agentA een buur-agentB tegenkomt in hetzelfde vakje van het grid, geeft de see functie door welke actie er uitgevoerd moet worden. Dit wordt doorgegeven aan de *act* functie.

3. 3 *Act functie*
De action die wordt uitgevoerd is geïmplementeerd in de act functie. Binnen onze simulatie zijn er twee acties; het weggeven van money en het bewegen binnen het environment. Op basis van de see functie wordt bepaald welke actie er wordt uitgevoerd. Zijn er buur agents en is agentA in bezit van money? Dan moet money worden weggegeven. Zijn er geen buur agents? Beweeg agentA dan naar een nieuwe positie.

3. 4 Update functie
De functie update dient voor het aanpassen van de interne staat van de agent. Wanneer de act functie is uitgevoerd moet de staat mogelijk worden geüpdatet, een agent die net zijn laatste money heeft weggegeven kan niet meer money weggeven. De staat moet van i0 naar i1 worden aangepast wanneer wealth = 0. Ook moet de update functie worden uitgevoerd wanneer een agent zich beweegt binnen de omgeving, hij krijgt dan een nieuwe positie.

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

• *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 wealth weg heeft gegeven dan wanneer hij na de vorige stap nog wealth 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 voor de agent veranderd.

• *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 wealth 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 neighbour heeft (zie Vb). 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.

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