1. Wat Unity ander maakt dan Netlogo is dat Unity gebouwd is voor een veel bredere scope aan mogelijkheden (vooral games), daarentegen is Netlogo specifiek gemaakt voor simulaties. Het verschil met Mesa is dat Unity een geheel eigen systeem is, waar Mesa een library is om te gebruiken in python, wat ook gelijk het volgende verschil is, namelijk programmeertaal. Unity maakt gebruik van C#, waar Mesa voor python is en Netlogo een compleet eigen taal heeft.

De voordelen van Unity zijn onder andere dat alles wat je gemaakt hebt zeer overzichtelijk is, mocht je natuurlijk alles vanaf het begin goed hebben gestructureerd (zoals mappen en objecten binnen een scene), maar ook dat je scripts heel simpel kan koppelen aan objecten (voordelig voor object oriented, gezien de 'this' nu ook gevisualiseerd is). Daarnaast is een voordeel van Unity dat er enorm veel keuze is in wat je kan gebruiken, niet alleen in de scene editor, maar ook qua extra functies voor in de scripts. Dit is echter ook gelijk een nadeel, want er zijn zo veel keuzes dat zonder tutorials of wekenlange studie er nauwelijks iets van te bakken is. Bij ieder probleem zal er vast een oplossing zijn, maar gezien de schaal van Unity moet daar waarschijnlijk lang op internet naar worden gezocht voor een bug vrij antwoord. Daarnaast is het debuggen moeilijk, gezien (in ieder geval bij gebruik van JetBrains Rider) de console in Unity zelf staat en na ieder teken dat erbij komt in het script moet dit eerst herladen. Ook geven sommige errors niet eens aan waar deze precies gebeuren. Dit alles maakt Unity erg tijdintensief.

Ik heb geprobeerd Reason en Act functies toe te voegen, inclusief een Automaton die tussen een random move (hier 0.02f in de x richting) en een beweging richting een Enemy kan wisselen. Echter is het niet gelukt om de wissel van states goed in de code te krijgen (dit begon al bij een huidige staat invoegen, waarna de getnextstate ook nog fout ging), waardoor de agent nu enkel nog iedere tick 0.02f in de x richting beweegt.

2. In het algemeen kan een agent als volgt beschreven worden:
    1. Een startstaat die onderdeel is van alle mogelijke staten.
    2. Een functie die de het blikveld vormt, waarin alle (staten van de) objecten tot een staat van de perceptie van de agent wordt gevormd.
    3. Een functie die de staat van de perceptie omzet in een actie de de agent uitvoert.
    4. Een functie die de huidige staat en staat van de perceptie omzet naar een volgende staat.

In de eigen machine kan de agent als volgt beschreven worden:
1. De startstaat van de agent (waarin de agent in een "random" richting beweegt)
2. Het script 'Sight.cs', dat de richting van de agent neemt en de hoek van het blikveld, waarna alles dat binnen het blikveld staat én de raycasts niet van worden geblokkeerd door andere objecten aan de 'inSight' array toevoegt. Hiernaast kunnen ook 'Smell', 'Touch' en 'Sound' aan worden toegevoegd
3. De functie 'MakeMove', die aan de hand van de nieuwe staat de actie (bijv. beweging) uitvoert
4. De functie 'Reason', dat met de argumenten verkregen bij sight (en alle andere mogelijke zintuigen) een volgende staat kan bepalen

3. **accessible** - Alles kan op zich in een agent worden gezet waardoor deze dus tot alle info toegang heeft. Dit was in de tutorial het geval totdat de Sight raycast werd gefilterd waardoor enkel de informatie van de kant die de agent op keek binnenkwam.
deterministic - Elke staat en actie kan zo geprogrammeerd worden dat deze altijd tot exact hetzelfde resultaat leidt. Natuurlijk kan in Unity ook met randomness worden gewerkt. Vooral output randomness kan tot een meer non-deterministic systeem leiden (mocht dit gewenst zijn).
episodic - De huidige actie van de agent heeft geen invloed op de toekomstige acties. Het zou wel mogelijk zijn om dit er in te programmeren
static - Unity zichzelf verandert, maar als alle instellingen en scripts van een agent hetzelfde blijven verandert daarbinnen niets
continuous - Zoals eerder gezegd, Unity verandert met iedere update. Unity kan op zich als discrete worden gezien binnen een update, aangezien er dan nog een eindig aantal mogelijkheden zijn (zowel met de editor als de code). In het grotere geheel kunnen er echter bij updates weer mogelijkheden bij komen, waardoor Unity als continuous kan worden gezien

4.