Naam: Sjoerd Beetsma  
Klas: V2A1  
Studentnummer: 6189293

1. **Volg de tutorial en omschrijf daarna in één paragraaf wat deze tool anders maakt dan andere programmeertalen, wat zijn de voor- en nadelen?**

Ik heb de tutorial van Stephen gevolgd in unity. Het is me naast het implementeren van de ‘cone collider’ gelukt om alles mee te doen. Echter vond ik de leer-curve van C# en Unity te lastig om zelf bedachte functionaliteit toe te passen voor de functies see, update en act.  
Echter heb ik met mijn groepje voor de eerste opdracht over Discrete Event Systems een multi-agent simulatie gemaakt waar een vos achter een konijn aangaat om het konijn te eten. En een konijn die probeert te vluchten van de vos. Graag gebruik ik deze code voor een aantal voorbeelden bij de volgende opdrachten. [sim-prog-2/sim.py at main · m-jeu/sim-prog-2 (github.com)](https://github.com/m-jeu/sim-prog-2/blob/main/sm2/lib/sim.py)  
  
Wat Unity anders maakt dan andere progammeertalen is ten eerste dat Unity geen progammeertaal is maar een tool.  
  
Unity bevat drie onderdelen:  
- Game engine  
- Applicatie/interface  
- Ingebouwde IDE  
  
Met Unity kan zowel 2D als 3D simulaties worden gemaakt. Vooral 3D simulaties vereisen een hoop implementatie werk als hier niet gebruik wordt gemaakt van een tool als Unity die een ingebouwde 3D (physics) engine heeft.  
Unity gebruikt als voornaamste programmeertaal om simulaties (games) in te maken C#. Voordelen van Unity zijn dat er erg veel voorgebouwde functionaliteit is met dank aan de game engine, zo hoef je zelf niet een physics engine te implementeren. Hierdoor kan je gelijk beginnen met je simulatie te bouwen in een simpele omgeving. Om dit in een pure programmeertaal te doen zoals python zou er veel meer werk gedaan moeten worden om eerst een simulatie framework te bouwen.  
Dit is ook gelijk een nadeel van Unity, je gebruikt veel bestaande functionaliteiten. Daardoor heb je minder goed overzicht over hoe elk van deze functionaliteit daadwerkelijk werkt ten opzichte van als je ze zelf van scratch zou implementeren.

1. **We definieren een staat en drie functies waarmee we een stateful agent abstract omschrijven:**
   * **Een initiele staat i0∈I, waarbij *I*alle mogelijk interne staten van de agent zijn**

De initiele staat i0 is de staat waarin de agent zich bevindt voordat de agent nieuwe percepties of acties ondergaat.  
In de tutorial van Unity is het niet gelukt om met states te werken.

* + **Een functie "See" of "Perceive", die een mapping maakt van elke staat in de omgeving tot een staat die de perceptie van de agent** **van de omgeving aangeeft. Dus: *See:* S⟶P, waar *S* de staat of serie van staten van de omgeving is en *P* de perceptie van die omgeving**

Een functie See of Perceive, in de unity tutorial de klasse ‘Sight’.  
Deze functie krijgt de staten van de wereld binnen (S), en geeft een perceptie (P) terug. De perceptie (P) is wat de agent als perceptie heeft van de wereld. Bijvoorbeeld in de Unity tutorial heeft de agent toegang tot de wereld via de klasse sight daarmee alle staten (S), echter kan de agent alleen in een X radius van zich af zien. De perceptie (P) van de agent die zijn dan alle objecten binnen een X radius van de agent. In de tutorial van unity is de perceptie van de agent bepaald aan de hand van een overlapSphere. Alles in de gehele wereld wat ook in de overlapSphere valt is de perceptie van de agent.

* + **Een functie "Act" die een kijkt naar de staat van de agent en een toepasselijke actie kiest. Dus *Act:* I⟶A, waar *I* een interne staat is en *A* een actie.**

Functie act die nog niet geïmplementeerd is na de tutorial.  
Deze functie krijgt een staat I en doet op basis hiervan een actie A.   
Een voorbeeld uit de DES-opdracht simulatie:  
Een konijn komt in de staat “flee” (I) terecht en onderneemt hierdoor ook de actie flee (A).

* + **Een functie "Update" die een staat *I* neemt (soms *D* genoemd) en perceptie *P*, en een nieuwe staat *I*oplevert, dus *Update:* I×P⟶I  
    Dit is een korte samenvatting van wat er staat in: "An introduction to Multi-Agent Systems" chapter 1.1 up to 1.4.  Die paragrafen kun je**[**hier**](https://canvas.hu.nl/courses/27428/files/2223184/download)**vinden.  
    Natuurlijk worden deze functies stukken complexer wanneer je agenten complexer worden.**

Een functie update neemt de huidige staat van een agent en zijn perceptie en kiest op basis hiervan een nieuwe staat.  
Bijvoorbeeld DES-opdracht simulatie:  
“Wander” (I) x Perceptie “Fox in view” (P) -> “Flee”, het konijntje zit in de staat “Wander” en heeft de perceptie “Fox in view”, resulterend in de staat “Flee”. Deze resulterende staat is wat de Functie act gebruikt om een actie uit te voeren.  
  
**Bijvoorbeeld: We hebben een simulatie met bankjes bomen en agents. De agents hebben een interne staat en die bestaat onder andere uit of ze op een bankje zitten,  of onder een boom zitten; ze kunnen alleen andere agents zien als samen op een bankje, en dan willen ook nog eens praten. De initiele interne staat van de agent bestaat dus uit hun positie en of ze, al dan niet alleen, op een bank of onder een boom zitten of niet. De functie See neemt een staat van de omgeving (dit is dus anders dan de interne staat van de agent) en gooit een percept eruit (zitten ze nu, al dan niet alleen, op een bank of onder een boom?) Ze zien alleen agents als ze samen meer dan 1 tick op een bankje spenderen. Als de agents op een bankje zitten blijven ze daar een tijdje zitten, en als ze er dan nog een agent komt te zitten zien ze die wel en gaan ze praten. Oftewel, de Act functie kijkt naar de interne staat van de agent. Als die beschrijft dat ze minder dan x lang op het bankje zitten, blijven ze zitten en als er nog een agent zit gaan ze daarmee praten. Dit verhaal is niet helemaal compleet natuurlijk, maar dan hebben jullie een idee van waar je aan kan denken.  
  
Beschrijf in eigen worden wat elk van de 4 concepten in het algemeen beschrijft EN wat het in jouw specifieke simulatie betekent.**

2.1:

1. **Beschrijf je omgeving op basis van de dichotomiën die**[**hier**](https://canvas.hu.nl/courses/27428/files/2223184/download)**op pagina 6 beschreven staan, en licht toe (dus niet alleen termen opsommen):**
   * **Accessible vs inaccessible**  
     Een accessible omgeving is wanneer de agent volledig ‘bewust’ is van de omgeving. Een voorbeeld hiervan is een schaakspel, de agents (spelers) weten op elk moment in het spel waar elk schaakstuk staat. Geen staat in de wereld is onbekend voor de agent. Een inaccesible omgeving is een omgeving waar de agent niet een perceptie heeft van de gehele wereld. Dit is ook de instantie in het de gemaakte simulatie van de FSM opdracht en ook in het voorbeeld van Unity. In beide simulaties hebben de agents een beperkte perceptie van de wereld gebasseerd op hun zicht sensoren.
   * **Deterministic vs non-Deterministic (Stochastic)**  
     De tutorial en eerder gemaakte opdracht zijn beide een deterministische omgevingen. In een deterministische omgeving is geen sprake van **kans/willekeur**. Als een agent precies dezelfde acties onderneemt in een deterministische omgeving zal het resultaat altijd hetzelfde zijn. Een niet-deterministische omgeving is een omgeving waar sprake is van willekeur. Bijvoorbeeld als er een kans is dat er bliksem neerslaat in de simulatie kan het zijn dat de Agent in één geval wel wordt geraakt door bliksem maar in een ander geval met precies dezelfde voorgaande acties niet.
   * **Episodic vs non-episodic (Sequential)**  
     Een episodic omgeving houdt in dat acties alleen gebasseerd zijn op huidige gebeurtenissen in de simulatie.  
     Non-episodic is een omgeving waar de geschiedenis invloed heeft op de huidige omgeving. Een voorbeeld hiervan is als er honger wordt gesimuleerd. Dit doordat er in het verleden niet is gegeten en wordt de staat dus beïnvloed door het verleden. In episodic omgeving hoeft de Agent dus ook geen rekening te houden met acties die mogelijk invloed zouden hebben op toekomstige states.
   * **Static vs Dynamic**  
     Een dynamische omgeving is een omgeving waar dingen gebeuren buiten de controle van de agent om.  
       
     Een statische omgeving is een omgeving waar de omgeving buiten de acties van de Agent op de omgeving, niet veranderd.  
     De tutorial van unity is statisch, de omgeving veranderd niet. Een statische omgeving veranderd niet naast wat de agent uitoefent op deze omgeving. Een voorbeeld van een dynamische omgeving is bijvoorbeeld een meer die bevriest in een gesimuleerde wereld. De omgeving veranderd, zonder invloed van de agent.
   * **Discrete vs continuous**Een discrete omgeving is een omgeving waar een beperkt aantal acties en percepties kan worden ondervonden. Zoals een bordspel als schaken, de agent(speler) kan zich beperken tot een beperkt aantal acties en is ook gelimiteerd tot alle mogelijke opstellingen van het bord. In het geval van een continue omgeving zijn er oneindig veel percepties en of acties.  
     In het geval van de tutorial is de simulatie continue de agent kan theoretisch oneindig veel enemies zien.
2. **Bedenk een voorbeeld waarbij minimaal 3 dichotomies precies** tegenovergesteld zijn en beschrijf of het veranderen van je omgeving op deze manier wel of niet iets zou toevoegen aan je simulatie.  
   Bijvoorbeeld: in een verkeerssimulatie maakt het soms wel echt uit voor de resultaten van je simulatie of de omgeving accessible is, maar bij het Nägelschreckenberg-model niet. Probeer te bedenken wat het effect van dit soort keuzes wat betreft je omgeving is op wat je kan leren/wat voor resultaten er uit je simulatie komen  
     
   Schaakspel Zelfrijdende auto  
   Accesible Inaccesible  
   Deterministisch niet-Deterministisch  
   Discrete Continue