Høgskolen i Ålesund

HØGSKOLEN I ÅLESUND

Aalesund University College

Agentbasert modellering

Undervisningsopplegg til faget Intelligente Systemer 2014

Modul 3. Intelligente Agenter
Modul 4: Intelligente systemer og Sosiale agenter

Av Harald Yndestad

Innhold

Forord

Dette dokumentet er et undervisningsopplegg i emnet intelligente agenter. Emnet er delt inn er i følgende hovedtema.

1. Starte med agent: Hvordan opprette enkle agenter
2. Partikkelagenter: Hvordan utvikle agenter som oppfører seg som partikler
3. Sosiale agenter: Hvordan agenter samarbeider i grupper

Opplegg

Undervisningsopplegget er bygget opp som begynnerkurs med en sum av leksjoner. I hver leksjon er det satt opp et sett med læringsmål, systemteori, framgangsmåte og oppgaver. Systemteori danner utgangspunktet for teorien om agentbasert modellering. Framgangsmåte beskriver en typisk framgangsmåte for modellering og programmering av agenter. Oppgave beskriver øvelsen for å oppfylle de forventede læringsmål. Hvert emne har en framgangsmåte som beskriver hvordan teorien om agentbasert modellering kan anvendes. Vi skal starte med å utvikle et enkelt rammeverk for modellering, før det gradvis innføres mer komplekse modeller.

Anvendelse

Dokumentet kan benyttes på flere måter. Du kan følge opplegget fra start og så gå gjennom alle leksjonene. Etter hvert kan du velge din egen rekkefølge eller du kan bruke enkelte leksjoner som et utgangspunkt for dine egne undersøkelser.

1 Starte med agenter

Innledning

Spørsmål: Hva er en agent?
Svar: En agent ufører en tjeneste på vegne av andre. Det innebærer i praksis at tjenesten har fått et delegert ansvar.

Spørsmål: Hva er da forskjellen på objekt og agenter?
Svar: Forskjellen ligger i at objekter utfører transformasjoner på data via algoritmer. Agenter er målsøkende objekter.

Spørsmål: Er ikke dette det samme?
Svar: Nei. Ikke helt. Objekter er basert på et deterministisk paradigme og kan sammenliknes med maskiner som produserer forutsigbare resultater. Agenter er basert på et statistisk paradigme med fri vilje til å optimalisere en tjeneste.

Spørsmål: Er der en standard metode for å modellere agenter?
Svar: Nei. Egentlig ikke. Agentbasert modellering kan være så mangt. Går en igjennom litteraturen om agenter, ser en at det gjøres forsøk med å utvikle standard programvare for agentbasert modellering. Samtidig ser en at dette er knyttet opp til spesielle anvendelser.

Spørsmål: Hva med et standard rammeverk?
Svar: Når du skal programmere agenter kan det være greit å ta utgangspunkt i et standard rammeverk for modellering og programmering av agenter. I dette undervisningsopplegget har jeg valgt å ta utgangspunkt i generell systemteori for modellering av agenter. Dette valget for å kunne benytte en mest mulig generisk teori for modellering av alle typer agenter. Med dette utgangspunktet, skal vi først lage oss ett rammeverk som vi så kan utvikle videre trinn for trinn, fram til en agent som utfører mer kompliserte oppgaver. I tillegg kan det være greit å benytte en standard skript rammeverk som benyttes som en mal for programmering av agenter.

Spørsmål: Hvordan kan vi lage et rammeverk for agent programmering?
Svar: En grei metode er å starte med et rammeverk for sanntidsmekanismene til agentene. Når dette er på plass kan det være greit å få etablert en agent server som oppretter og forvalter en gruppe med agenter. Agentens tjenester kan da utvikles gradvis etter behov. Det er da vanlig å starte med en Hallo Agent.

Læringsmål

En skal ved avslutning av denne leksjonen.

Kunnskaper

1. Kunne redegjøre for forskjellen mellom objekter og agenter

Ferdigheter

1. Kunne opprette en Hallo Agent som et rammeverk
2. Kunne opprette visuelle Agenter
3. Kunne opprette et 3D landskap i Unity spillmotor
4. Kunne importere en terrengmodell til Unity spillmotor
5. Kunne benytte basistjenester i Unity spillmotor
6. Kunne opprette et rammeverk for en agentbasert modellering

Generelle kunnskaper

1. Om agent rammeverk
2. Om Unity spillmaskin som teknologisk plattform

TEMA

1. Agent rammeverkt
2. Hallo agent og visuelle agenter
3. Sanntidsstyring av agent tilstand
4. Rapportering av agent tilstander
5. Tekstur, sjø, sol, kamera og bakgrunn til landskap
6. Unity som teknologisk plattform.

Teknologisk plattform

Før du begynner å programmere agenter, bør du tenke gjennom hvilken teknologisk plattform og programmeringsspråk du skal benytte.

Spørsmål: Kan en bruke Matlab til programmering av agenter?
Svar: Svaret er ja. I Matlab kan du representere landskap og agenter i tabeller. Landskapet modelleres som et 2D array. Agenter egenskaper kan lagres i en vektor, som igjen lagres i en liste. Bruker du siste versjon av Matlab, kan agentene programmeres som objekter, og lagres i lister. Fordelen ved å benytte Matlab, er at du har direkte tilgang til andre Matlab tjenester som GA, Neurale nettverk osv. Det kan derfor ofte være nyttig å starte med Matlab når du skal prøve ut en algoritme. Ulempen er at Matlab mangler visuelle framstilling av agenter og landskap, og mange av de sanntids tjenester en finner i en spillmotor.

Spørsmål: Kan en bruke Java til programmering av agenter?
Svar: Svaret er ja. Java er et objektorientert språk. Dette forenkler betydelig programmeringsarbeidet. I Java er landskapet representert som et 2D array. Agenter programmeres som objekter og lagres i lister. Fordelen ved å benytte Java, kan være at du har mer kunnskaper om dette språket. Ulempen er at Java mangler de visuelle framstillinger og de sanntids tjenester en finner i en spillmotor.

Spørsmål: Hvorfor bruke Unity spillmotor?
Svar: Unity brukes av mer enn 250.000 spillprogrammerere og regnes som en av de beste spillmotorene på markedet. Landskap programmeres som 2D array. Agenter er programmert som visuelle agenter og lagres i lister. Fordelen med å bruke Unity spillmotor, er at en får en visuell framstilling av agentene. I tillegg har spillmotoren et godt utviklet rammeverk for kontroll av sanntidsprosesser og for kontroll av interaksjon mellom agentene. Dette gjør at Unity er spesielt egnet til modellering av agenter, spill og modellering av komplekse systemer. Ulempen med Unity er at en må sette seg inn en teknologisk plattform, som i realiteten er et utviklingssystem.

Unity kan programmeres i flere språk. De viktigste er JavaScript og C#. JavaScript er et enkelt programmering språk. Det fleste Tutorials benytter derfor JavaScript i sine eksempler. Ulempen er mangel på standardisering. I dette dokumentet har jeg valg å benytte C#, siden dette ligger nærmere opp til Java programkode.

Spørsmål: Hva trenger en å lære seg, om en skal benytte Unity spillmotor?
Svar: Unity er et integrert utviklingssystem. Det er lett å komme i gang å få til noe

som virker og det er lett å lage et komplisert landskap. Samtidig er det stort utviklingssystem med rom for utvikling av komplekse systemer.

En farbar framgangsmåte er at du:

1. Starter med å laste ned en 30 dagers gratisversjon. Se litt på Island Demo. Da får de en følelse at mulighetene som ligger i Unity som utviklingsverktøy.
2. Ta en gjennomgang av video Tutorials. Da ser du hvordan du kan bygge opp en scene.
3. Bygg opp noen enkle scener selv via et par leksjoner. Da får du programmeringen inn i fingrene. Prøv f.eks Fireball og "Hello World in 6 minutes". Disse inneholder det meste du trenger.
4. Se spesielt på script programmering i Unity.
5. Når du begynner å programmere agenter, kan det være lurt å starte med en blank scene. Agenter visualiseres som en kule, og landskap som en flate.
6. Når ting begynner å virke, kan du etter hvert sette på mer kompliserte visuelle 3D scener og agenter, som pynter opp i landskapet.

Unity informasjon

Spørsmål: Hvor finner jeg informasjon om Unity?
Svar: Dersom du starter her, finner du det meste.

1. Unity http://unity3d.com
2. A Unity free 30 day trial here: http://unity3d.com/unity/download.
3. Unify Community Wiki (UCW): http://www.unifycommunity.com/wiki
4. Unity Forums: http://forum.unity3d.com
5. Unity Documentation: http://unity3d.com/support/documentation
6. Virtual More Wiki: http://virtualmore.org/wiki
7. Virtual More Blogg: http://www.virtualmore.org/blog/

Unity demos

Spørsmål: Finner jeg demo videoer om bruk av Unity på nettet?

Svar: Ja. Her finner du en god begynnelse

1. YouTube: http://www.youtube.com/watch?v=q3Cy0IAd6gk
2. Unity demos: http://unity3d.com/support/documentation/video/

Unity tutorials

Spørsmål: Er det tilstrekkelig å studere noen videoer?
Svar: Nei. Dessverre. Du må nok starte ved begynnelsen med praktisk øving, slik at du får det inn i fingrene. Du kan da begynne her.

1. Unity tutorials: http://unity3d.com/support/documentation/video/
2. VM i 6 minutes: http://www.youtube.com/watch?v=LP73jeWtaNY
3. Fireball: Notat av Robin Trulssen Bye.
4. Virtual More Tutorials: http://virtualmore.org/wiki/index.php?title=Tutorials

Video eksempler på partikkel agenter

1. Particles in a river: http://www.youtube.com/watch?v=qkZGcNyrlOk
2. Particle in a river: http://www.youtube.com/watch?v=iW0C4M4yY5U&feature=related

3. Massive particles:
   http://www.youtube.com/watch?v=7cjorOe810o&feature=related

Utstyr til oppgaven

1. Unity spillmotor: 30 dagers gratis prøveversjon finner du her:
   http://unity3d.com/unity/download.
2. Tilgang til en 3D-modell av et objekt. Gjerne utviklet i 3D verktøyet Blender, Maya eller i 3D Studio. Denne type modeller er lett tilgjengelig via internet.
3. Tilgang til en 3D modell av et kart

Litteratur

Spørsmål: Er der lærebøker om Unity?
Svar: Ja. Det er nå fen rekke eBbøker om Unity på markedet. Se: www.PacktPub.com

1. Will Goldstone: Unity Game Development Essentials. PacktPub

1.1 Hallo Agent

Spørsmål: Hva er en Hallo Agent?
Svar: Hallo Agent er din første agent. Dette er en god begynnelse. Tjenesten til en Hallo Agenter er å rapportere en enkel Hallo melding tilstand til en bruker. Dette er første trinn i utvikling og uttesting av agentens egenskaper.

Spørsmål: Hvordan kan vi lage en enkel Hallo Agent?
Svar: Det enkleste er å lage metode ReadMe(). Denne metoden kan du sette inn i Agent scriptet. Med enkle endringer kan denne enkle metoden brukes senere til nye oppgaver. Etter hvert vil du også ta i bruk andre metoder for å kommunisere med agenten. Når det blir mange data å holde rede på, kan du lese de globale variable i Unity inspector, eller en kan lage et GUI som avlese tilstandene i sann tid.

LÆRINGSMÅL

En skal ved avslutning av denne leksjonen ha følgende

Kunnskaper

1. Kunne opprette en Hallo agent som rapporterer sin egen tilstand
2. Kunne opprette et landskap på en spillmotor

Ferdigheter

1. Kunne benytte basistjenester i Unity spillmaskin
2. Kunne opprette et rammeverk for en agentbasert modellering

Generelle kunnskaper

1. Om agent rammeverk
2. Om Unity spillmaskin som teknologisk plattform

TEMA

1. Rapportering av agent tilstand
2. Sanntidsstyring av agent tilstand

SYSTEMTEORI

Hallo Agent har en system arkitektur med bare en tjeneste

A(arc, t) = {Output}

der Output gir en enkel tilstandsrapport til omgivelsen. I dette tilfellet er dette et objekt som rapporterer ut en enkel melding til bruker. Denne enkle tjenesten er en god start til å utvikle en mer fullverdig agent.

FRAMGANGSMÅTE

Modellering

1. Sett opp en enkel systemmodell for agenten
2. Formuler agentens metoder for utførelse av tjeneste.

Programmering

1. Start opp en nytt prosjekt i Unity
2. Opprett en ny katalog ”myScenes” under ”Project”. Velg ”File” ”Save Scene”, gi scenen navnet HelloAgentScene, og flytt filen til ”myScenes.
3. Opprett en C#-file under ”Project” og skriv inn vedlagte script for ”HelloAgent”. Skriv om filnavnet til ”HelloAgent” og legg filen inn under en ny katalog ”myScripts”. Se vedlagte skript eksempel-
4. Velg ”GameObject” ”Create Other” ”Sphere”. “Sphere” under ”Hierarchy” skrives om til “AgentObject”.
5. Dra scriptet ”HelloAgent” til ”AgentObject” under “Hierarchy”.
6. Velg “HelloAgentScene” og start programmet via piltasten. Meldingen fra agenten blir da skrevet ut nederst til venstre.

Verifisering

1. Test om agenten utøver den ønskede tjeneste

OPPGAVE

Kontrollspørsmål om intelligente agenter

1. Hvilke tjenester har en Hallo Agent?
2. Hvilket rammeverk er knyttet til programmeringen av en Hallo Agent
3. Hvordan kan en Hallo Agent benyttes videre til programmering av agenter?

Forsøk med Hallo-agenter

1. Opprett en HalloAgent som beskrevet ovenfor. Modifiser agentens programkode slik at den skriver ut agentens nummer hvert sekund
2. Opprett en ny metode newNumber() som setter inn et tilfeldig agent nummer.

1.2 Visuell agent

Spørsmål: Hva er en visuell agent?
Svar: En visuell agent er en agent med 3D framstilling av agentens visuelle egenskaper.

Spørsmål: Hva er nytteverdien med visuelle agenter?
Svar: Nytten av visuelle agenter er å få fram en visuell representasjon som representerer agenten. Ved modellering av enkle systemer kan innføring av

visuelle agenter lett bli ekstra plunder og heft. Det er da enklest å framstille agenter som små kuler. Når du utvikler mer kompliserte scener vil du gjerne ha en bedre oversikt over det agentene representerer. En visuell agent som representerer agentens egenskaper kan da være en og god ide. Dette er vanlig i databaserte spill.

Spørsmål: Hvordan kan vi få fram en god visuell representasjon av en agent?
Svar: Det kan en gjøre ved å knytte en visuell 3D-modell direkte til agenten i spillmotoren.

LÆRINGSMÅL

En skal ved avslutning av denne leksjonen.

Kunnskaper

1. Kunne gjøre rede for anvendelse av visuelle agenter

Ferdigheter

1. Kunne opprette visuelle Agenter
2. Kunne benytte basistjenester i Unity spillmaskin
3. Kunne opprette et rammeverk for en agentbasert modellering

Generelle kunnskaper

1. Om visuelle agenter i spill og 3D modellering.
2. Om Unity spillmaskin som teknologisk plattform.

TEMA

1. Visuell hallo agent
2. Sanntidsstyring av agent tilstand
3. Rapportering av agent tilstand

SYSTEMTEORI

Visuelle agenter har en kropp som representerer agentens visuelle utforming. I dette tilfellet har agenten ingen andre funksjoner. Den kan formuleres med den enkle system arkitekturen

A(arc,t) = {Body}

der Body representerer agentens visuelle egenskaper i landskapet.

FRAMGANGSMÅTE

Modellering

1. Ta utgangspunkt i leksjonen Hello Agent

Programmering

1. Oppretting av 3D objekt: Opprett en visuell 3D-modell av et objekt i et 3D modelleringsverktøy. Bruke du Maya, Blender, eller 3D Studio, kan objektet importeres direkte inn i Unity.
2. Import av 3D objekt: Velg *Assets* og *Import New Asset.* Deretter går du inn i en katalog og velger 3D-objekt, og last inn 3D objektet. Objektet blir nå liggende som et prefab objekt i prosjekt katalogen.

3. Katalog med 3D objekter: Oppretter en katalog Resources under Prosjekt, og flytt objektet inn under katalogen Resources.
4. Skalering: Dra det visuelle objektet inn i scenen og tilpasser ønskede skalering, overflate og liknende under vinduet ”Inspector”.
5. Visuell Agent: Dra den importerte 3D modeller inn til AgentObject under Heierarchy. Du vil nå se 3D objektet i Scenen og i Game vindu. Du kan nå lage kopier av agenten ved å dra nye kopier av prefaben inn i scenen.

Verifisering
1. Sjekk om utskriften blir som forventet
2. Sett inn rapporteringer i agenten dersom der er avvik fra det du forventet

OPPGAVE
Kontrollspørsmål om visuelle agenter
1. Hva er en Visuell Agent
2. Hvorfor kan det være nyttig at agenter har en visuell representasjon?
3. Hvor er det visuelle agenter helst blir benyttetP

Forsøk med visuelle agenter
I denne oppgaven kan vi starte med å utvikle en enkel visuell 3D representasjon.
1. Modellering: Velg en ferdig 3D modell av et objekt, utviklet med et 3D modellering verktøy som Blender, 3D studio eller Maya.
2. Programmering: Opprett en visuell agent som beskrevet ovenfor.
3. Diskusjon: Vil kopier av visuelle agenter produsere samme agent nummer?

1.2 3D Landskap
Spørsmål: Hva er et 3D landskap?
Svar: Et 3D landskap er det samme som en terrengmodell. Landskap er stedbunden informasjon utenfor agenten. Agenter kan normalt avlese informasjon i landskap og samtidig påvirke informasjon i landskap. Det er da slik at agenten må ha tilgang til et tilrettelagt landskap, samtidig som den må kunne prege landskapet.

Spørsmål: Hvilken nytte har bruken landskap?
Svar: Modellering av landskap er en metode til å organisere stedbunden informasjon som kommuniserer med agenter. Når landskap får en visuell framstilling, er det lett å identifisere komplekse sammenhenger.

Spørsmål: Hvordan kan vi lage et 3D landskap som en terrengmodell?
Svar: Du kan lage et landskap som en 3D objekt i Unity. Dette kan så utformes manuelt som en terrengmodell og så tilføres tekstur, slik at det ser pent ut.

LÆRINGSMÅL
En skal ved avslutning av denne leksjonen
Kunnskaper
1. Kunne gjøre rede for metoder for utvikling av 3D landskap på spillmotor

Ferdigheter
1. Kunne sette tekstur, sjø og bakgrunn til et landskap

2. Kunne opprette manuelt et 3D landskap på en spillmotor
3. Kunne importere en terrengmodell
4. Kunne pynte landskapet til en mer realistisk visuell scene

Generelle kunnskaper

1. Kunne benytte basistjenester i Unity spillmaskin
2. Ha kjennskap til mulighetene til å utvikle 3D landskap i spillmaskiner

TEMA

1. Manuell modellering av terreng
2. Omport av 3D terrengmodell
3. Tekstur, vegetasjon, sjø, sol, kamera og bakgrunn

SYSTEMTEORI

Agenter i landskap er en del av metasystemet

$$S(t)=\{N(t), A(t), L(t)\}$$

Der L(t) representerer landskapet, A(t) agenten og N(t) agentens relasjon til landskapet.

FRAMGANGSMÅTE

Manuell oppretting av 3D terrengmodell

1. Skape et terreng: Velg *Terrain* og *Create Terrain*
2. Justere terreng: Juster terrengets vinkel i forhold til kamera.
3. Set høyde: Velg *Raise High* under Inspector og vel en passende skalering for å endre kartet. La pilen ”male” på plass det terrenget du vil ha. Snu så terrenget til en naturlig vinkel

Import av høydekart

1. Høydekart: Du tar utgangspunkt i et ferdig høydekart på DEM-format.
2. Import: Du importerer høydekartet fra en katalog ved å velge *Terrain* og *Create Terrain* og igjen *Terrain* og *Import Hightmap*. Deretter velger du den lagrede kartfilen.
3. Skalering: I vinduer *Import hightmap* velger du om plattformen er Mac eller PC. Velg *Terrain Size* og sett skaleringene til kartet (Ålesund: 15500x, 1100y, 15500z). Velg så import.

Tekstur til terrengmodell

1. Velge tekstur: Velg Terrain objektet. Velg en malingskost under *inspector*. Velg så *Edit texture* og *Add texture*. Du får da opp en liste med mulige teksturer.
2. Type: Velg skalering på tekstur som samsvarer med høydekartets oppløsning (f.eks 1550x15500m), og deretter *Add*.

Vegetasjon til terrengmodell

1. Velge tekstur: Velg Terrain objektet. Velg en malingskost under *inspector*. Velg så *Edit gras* og *Add gras*. Du får da opp en liste med mulige gress eller vegitasjon typer.
2. Type vegetasjon: Velg type gres, tre eller plante.

3. Setting av vegitasjon: Vegetasjon kan så "males" på terrenget med penselen.

Sjøoverflate til terrengmodell

1. Velg vann tekstur: Gå inn i *Projects* katalogen under *Standard assets*. Her finner du prefaben *Daylight water*.
2. Skalering: Dra prefaben *Daylight water* inn i terrenget. Juster så høyde, posisjon og skalering av prefaben *Daylight water.*

Himmelbakgrunn til terrengmodell

1. Velg tekstur: Velg *Edit* og *Render Settings*. Velg *Blue Sky* i *Inspector*.
2. Framvisning: Trykk på *Scene iconed* under *Scene* øverst til venstre.

Lys til terrengmodell

1. Velg lyskilde: Velg *GameObject. Create Other* og *Direct Lighet*. Du får da fram en sol
2. Orientering: Skaler solens posisjon og retning.
3. Lyskart: Etter at solen har funnet en naturlig stilling lager du er standard lyskart slik at sol og skygger settes inn i terrenget. Dette gjør du ved å velge *Terrain* og *Create Lightmap*, Merk av skygge dersom du vil ha denne med.

OPPGAVE

Kontrollspørsmål om agenter i landskap

1. Hva menes med et Agent landskap
2. Hva er det agentbaserte landskap kan representere?
3. Hvordan er koblingen mellom agenter og landskap?

Forsøk med agenter i landskap

I denne oppgaven skal det utvikles et et visuelt 3D landskap i Unity spillmotor.

1. Studer videoen VM in 6 minutes: http://www.youtube.com/watch?v=LP73jeWtaNY
2. Terrengmodell: Importer terrengmodell for Møre
3. Tekstur: Legg på terreget, tekstur, sjø, sol og bakgrunn
4. Diskusjon: Hva tror du er mulighetene og begrensningene ved 3D modellering av landskap?

2 Generell systemteori

Spørsmål: Finnes det en etablert metode for modellering av agenter?
Svar: Neppe. Agentbasert modellering er forholdsvis nytt tema. Det er ennå ikke er satt seg en etablert metode for agentbasert modellering, slik en finner i f.eks objektorientert programmering.

Spørsmål: Hvilken metode benyttes i dette undervisningsopplegget?
Svar: I dette undervisningsopplegget er det valgt å ta utgangspunkt i generell systemteori, utviklet med prøving og feiling over flere år.

Spørsmål: Hva er fordelen med denne metoden?
Svar: Det er en generisk metode som lett kan benyttes ved alle typer agentbaserte modelleringsoppgaver.

LÆRINGSMÅL

Studenten skal ved avslutningen av denne leksjonen kunne:

1. Gjøre rede for begreper og metoder for modellering av agenter basert på generell systemteori
2. Anvende systemteori til agentbaserte modellering
3. Ha generelle kunnskaper om systemteori
4. Ha generelle kunnskaper om agentbasert modellering

SYSTEMTEORI

Et system S(t) er sammensatt av et sett partnere P(t), som samarbeider om et felles formål, der

$$S(t)=\{B(t), s(t)\}$$

Der B(t) representerer et sett med relasjoner mellom partnere P(t), og hver partner element s(t), representerer et sett delsystemer. System modeller kan neddeles til de duale egenskapene

$$S(\text{formål}, t) = \{S(\text{otologi}, t), S(\text{kunnskap}, t)\}$$
$$S(\text{ontologi}, t) = \{S(\text{arkitektur}, t), S(\text{dynamikk}, t)\}$$
$$S(\text{kunnskap}, t) = \{S(\text{etikk}, t), S(\text{læring}, t)\}$$
$$S(\text{etikk}, t) = \{S(\text{formål}, t), S(\text{begrensning}, t)\}$$
$$S(\text{læring}, t) = \{S(\text{identifikasjon}, t), S(\text{kontroll}, t)\}$$

System modeller S(t) kan representere alle typer at biologisk, sosiale, teknologiske og abstrakte organisasjoner.

Agentbasert systemmodell

Agentbaserte modeller A(t) er organisasjoner knyttet opp til landskap L(t). En agentbasert systemmodell basert på agenter kan da formuleres som:

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett agenter, L(t) er sett med landskap og N(t) er sett med relasjoner mellom agenter og landskap.

Gjensidig kobling

Der er en gjensidig kobling mellom system elementer der

$$A(t) = f(L(t), N(t)),\ L(t) = f(A(t), N(t)),\ N(t) = f(A(t), L(t))$$

Det vil si at agenter påvirkes av landskap, og landskap påvirkes av agenter. Agentens nettverk N(t) representerer relasjoner mellom agenter og mellom agenter og landskap.

INDIVIDBASERTE AGENTER

Agentens interne egenskaper kan modelleres med system perspektivene

$$S(\text{agent}, t)=\{A(\text{arc},t), A(\text{dyn},t), A(\text{eti},t), A(\text{lea},t)\}$$

der A(arc,t) representerer agentens arkitektur, A(dyn,t) agentens dynamikk, A(eti,t) agenten etikk og A(lea,t) agentens læring.

Agent arkitektur A(ark, t)
Agenter har en intern arkitektur som er sammensatt av et sett med tjenester:

A(ark, t)={Sensor, Body, Producer, Motor, Control, Output}

Agent(Sensor, t) representerer en del av agentens nettverk som måler tilstander i landskap og til andre agenter.

Agent(Body, t) representerer egenskaper ved agentens form. Det omfatter visuelle egenskaper, friksjon mot omgivelser og liknende.

Agent(Producer, t) er en tjeneste som utfører transformasjoner ved å konsumere noe og produsere noe. Det kan være en vare, en tjeneste, agentens ressurser, eller agentens rekruttering.

Agent(Motor, t) er tjenester som omsetter agentens ressurser til en bevegelse i landskapet.

Agent(Control, t) utfører kontrollfunksjoner i agenten.

Agent(Output, t) er en del av agentens nettverk som kan påvirke tilstanden i landskap eller i andre agenter.

Agent dynamikk
Agenter dynamikk har en intern og en ekstern referanse. Den interne agent dynamikk har egenskapene

A(dyn, intern, t)={Tilstand, Oppførsel, Mål}

Agent(dyn, tilstand, t) representerer agentens tilstands dynamikk. Eksempler på tilstandsdynamikk er dynamiske egenskaper i forhold til bevegelse, vekst, produksjon og liknende.

Agent(dyn, oppførsel, t) representerer agentens oppførsel dynamikk. Begrepet oppførsel er knyttet til rekkefølge eller faser i arbeidsoperasjoner for å nå mål.

Agent(dyn, mål, t) representerer endringer i agentens mål. Agenter er målsøkende objekter. Når mål også har en dynamikk, betyr der at agenter også har evne til å velge nye mål.

Agent etikk
Agenters etikk har egenskapene

A(eti,t) = {Mål, Begrensninger}

Det vil si at agenter er målsøkende av natur. Samtidig er det slik at mål relateres til agentens begrensninger.

Agentens etikk A(eti,t) har et internt og et eksternt perspektiv. Agentens interne etikk A(eti, int. t) søker å optimalisere agentens egne ressurser. Agentens eksterne etikk A(eti, eks. t) søker å tilpasse agentens mål og egenskaper til et formål med ekstern referanse. Det vil si at agenter representerer et system element.

Agent læring
Agent læring A(lea, t) er å tilpasse agentens tilstander slik at en realiserer agentens interne etikk A(eti, int, t) og eksterne etikk A(eti, eks, t).

Intern læring A(les, int, t) vil si at agenten lærer å optimalisere egne ressurser. Ekstern læring vil si at agenten lærer optimalisere egne kostnader til eksterne forhold. Det kan være å identifisere mål med ressurser, og å nå målene med minst mulig kostnader.

Yteevne
Intelligente agenter kan endre egenskaper ved å overvåke agentens egen yteevne. En metode til å overvåke agenten yteevne er å beregne en kostnadsfunksjon.

$$J(t) = Midlere(e(t)Qe(t), u(t)Ru(t))$$

Der e(t) representerer avvik fra mål, Q representerer kostnaden på avviket, u(t) representerer et eksternt pådrag og R representerer kostnadene ved det eksterne pådraget.

SOSIALE AGENTER
Sosiale agenter er et sett med agenter med felles formål. Et felles formål tilsier at sosiale agenter representerer delsystemer. De sosiale agenter er videre karakterisert med at de er underlagt et felles regelverk.

Sosial Agent arkitekturer
Et sett sosiale agenter danner til sammen en Sosial agent arkitektur. Den sosiale arkitektur sier noe om hvordan agenter er forbundet til hverandre via nettverket N(t). Typiske eksempler på Sosiale agent arkitekturer er:

A(sos,ark, t) = {gruppe, sverm, kø}

Sosial Agent dynamikk
Sosiale agenters dynamikk sier noe om hvordan gruppen utvikler seg over tid. Typiske eksempler på Sosiale agent dynamikk er:

A(sos,dyn, t) = {gruppe hastighet, gruppe hast retning, }

Sosial Agenter etikk
Sosiale agenters etikk sier noe om gruppens motiv for å samarbeide om et felles formål. Typiske eksempler på Sosiale agenters etikk er:

A(sos, eti, t) = {Identifisere ressurser, beite ressurser, redusere risiko}

Sosial Agent læring
Sosiale agenters læring sier noe om gruppens regler for å realisere den sosiale etikk. Typiske eksempler på Sosiale agenters læring er identifikasjon og kontroll av:

A(sos, lær, t) = {trussel, ressurser tilgang, risiko}

FRAMGANGSMÅTE
En typisk framgangsmåte ved utvikling av systemmodeller
1. Datafangst: Hente inn informasjon om temaet

2. Landskalp: Identifisere alle landskap til modeller
3. Agent grupper: Identifisere alle agent grupper på samme systemnivå
4. Nettverk: Identifisere systemmodellens nettverk
5. Formål: Formulere system modellens formål og etikk
6. Delsystemer: Formulere modellens delsystemer

OPPGAVE

Kontrollspørsmål om agentbaserte systemer

Om kunnskaper

1. Gjør dere for definisjonen på en systemmodell
2. Gjør rede for systemegenskapene til en agentbasert systemmodell

Om ferdigheter

Der skal utvikles en agentbasert systemmodell for industriell oppdrett

1. Formuler egenskapene til et typisk landskap som modellen
2. Identifiser typiske agent grupper i modellen
3. Formuler modellens formål og etikk
4. Formuler typiske regler for læring

3 Individbaserte Agenter

3.1 Passive partikkel agenter

Spørsmål: Hva er en passiv partikkel agent?
Svar: En passiv partikkel agent har ingen fri vilje til å tilpasse seg et eksternt miljø.

Spørsmål: Hva er eksempler på passive partikkel agenter?
Svar: Eksempler på passive agenter er partikler som representerer objekter eller materialer i fritt fall, eller som renner langs et landskap.

Spørsmål: Hva kan passive partikkel agenter i stasjonære landskap gjøre?
Svar: Typiske oppgaver er å identifisere komplekse strømningsmønster. F.eks strømningsmønster i vann, bølger osv.

LÆRINGSMÅL

Studenten skal ved avslutningen av denne leksjonen kunne:

1. Utvikle en systemmodell for partikkelagenter
2. Planlegge et eksperiment ved bruk av partikkelagenter
3. Kunne simulere en passiv partikkelagent i fritt fall
4. Opprette en passive sverm partikkelagenter

SYSTEMTEORI

Metasystemet for en enkel agent i et landskap kan formuleres som:

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer en agent, L(t) landskapet og N(t) er relasjoner mellom agent og landskap.

Nettverket har relasjonen N(t)={A(t), L(t)} der A(t) representerer en agent med stasjonær tilknytting mellom agent og landskap.

Vi tenker oss at landskapet L(t) er representert med en 3D matrise der en agent A(1,t) konsumerer ressurser fra cellen L(1,x,y) og produserer noe til 2D landskapet i cellen L(2,x,y).

I dette tilfellet er landskapet konstant. Det kan formuleres som:

$$L(x,y,z,t)=K(x,y,z)$$

der K(x,y,z) er en 3D representasjon som ikke endrer seg med tiden.

Agentmodell
Agenter kan modelleres med egenskapene

$$A(t)=\{A(ark,t),A(dyn,t),A(eti,t),L(lær,t)\}$$

Der A(ark,t) representerer agentens arkitektur, A(dyn,t) agenten tilstandsdynamikk, A(eti,t) agentens etikk, og L(lær,t) agentens læringsstrategi.

Agent arkitektur A(ark,t):
En typisk agent arkitektur A(ark,t) for en stedbundet agent vil være:

$$A(ark,t)=\{Sensor, Body, Motor\}$$

Der Sensor måler og oppdaterer tilstanden til landskapet L(z,x,y,t), Body representerer agentens form som en visuell agent. Motor er en tjenester som ivaretar agentens framdrift. Typiske tjenester for en partikkelagent vil være:

A(Sensor,t)={Avleser agentens nære omgivelse}.
A(Body,t)={Representerer partikkelens geometri og frikssjon}
A(Motor,t)={Ivaretar partikkelens framdrift}

Agent dynamikk A(dyn,t):
Agent dynamikk representerer agentens dynamiske egenskaper for hvordan den utvikler seg i tid. En typisk dynamisk modell for en partikkelmodell vil være:

Partikkel akselrasjon vektor: $a(part.,t) = a(part,t)+F(motor,t)/masse$
Partikkel kraftvektor: $F(motor,t) = -K(friksjon)v(part,t)+F(ekstern,t)$
Partikkel hastighet vektor: $v(part,t+T) =v(part,t)+T*a(part,t)$
Partikkel posisjon vektor: $x(part,t+T) = x(part,t)+T*v(part,t)$

Der T er samplingsintervallet.

Agent etikk A(eti,t):
Partikkelagenter har ingen fri vilje og har derfor heller ingen system etikk.

Agent læring A(lær,t)
Agent læring er realisering av agentens etikk. Når agentene har ingen etikk, har de heller ingen evne til læring.

FRAMGANGSMÅTE

Agentbasert simulering av passiv partikkelmodell kan betraktes som et laboratorium eksperiment der en tar utgangspunkt i følgende framgangsmåte.

1. Sett opp en meta systemmodell for eksperimentet
2. Sett opp en agentmodell
3. Formuler agentens formål og metoder
4. Programmer agentens egenskaper
5. Dokumenter agentens egenskaper
6. Diskuter dine resultater

Framgangsmåte for bruk av Unity fysikkmotor:

1. Opprett et stasjonært landskap som fanger opp fallende agenter
2. Ta utgangspunkt i HalloAgent og opprett en ny partikkel agent
3. Innfør gravitasjon for agenten i agentens fysikkmotor
4. Innfør en kollisjons boks rundt agenten
5. Studer hvordan en enkelt agent faller i landskapet
6. Opprett en AgentServer for oppretting av en gruppe agenter
7. Opprett en gruppe agenter der du tester ut bevegelsen til en samlet gruppe partikkel agenter

Framgangsmåte for bruk av partikkelmotor:

1. Slå av gravitasjon i Unity fysikkmotor
2. Opprett en metode AgentMotor() som påvirker agentens beregner agentens hastighetsvektor
3. Innfør gradvis nye tjenester i AgentMotor(). Prøv først en enkel agent der du innfører en gravitasjon. Deretter innfører du friksjon for å studere hva som skjer.
4. Opprett en AgentServer for oppretting av en gruppe agenter
5. Opprett en gruppe agenter der du tester ut bevegelsen til til en samlet gruppe partikkel agenter

OPPGAVE

Kontrollspørsmål om partikkel agenter

1. Gjør rede for hva som menes med en passiv partikkelagent.
2. Hva kan en passiv partikkelagent representere?
3. Hvordan kan vi lage en fysikkmodell for en passiv partikkelagent?

Forsøk med partikkel agenter

I dette eksperimentet skal vi studere hvordan en gruppe passive partikkel agenter, med masse, oppfører seg i et lukket landskap.

1. Opprett et tilfeldig lukket landskap og studer hvordan en gruppe passive partikkel agenter oppfører seg når agentene benytter gravitasjon fra Unity fysikkmotor.
2. Opprett et tilfeldig lukket landskap og studer hvordan en gruppe passive partikkel agenter oppfører seg når agentene benytter sin egen fysikkmodell.

3.2 Aktive partikkel agenter

Spørsmål: Hva er aktive partikkel agenter?

Svar: Aktive partikkel agenter er partikkel agenter med evne til å tilpasse seg en tilstand i landskapet Det betyr at agenten et tildelt fri vilje til tilpasse seg lokale forhold.

Spørsmål: Hva er eksempler på aktive partikkel agenter?
Svar: Eksempler på aktive partikkel agenter er partikler er agenter som representerer partikler eller objekter med en for oppdrift. Det kan f.eks være partikkel agenter som representeres fysiske eller organiske materialer eller vann eller luft. Det kan også være agenter som representerer biltrafikk, skip og liknende.

Spørsmål: Hva kan partikkel agenter i stasjonære landskap gjøre av oppgaver?
Svar: Typiske oppgaver er å identifisere komplekse strømningsmønster. F.eks strømningsmønster i vann, bølger osv.

LÆRINGSMÅL

Studenten skal ved avslutningen av denne leksjonen kunne:

1. Gjøre rede for anvendeleser av aktive partikkelagenter
2. Utvikle en systemmodell for en aktiv partikkelagenter
3. Planlegge et eksperiment ved bruk av aktive partikkelagenter
4. Opprette en sverm aktive partikkelagenter
5. Bevege partikkelagenter som påvirkes av dynamisk landskap

SYSTEMTEORI

Metasystemet for en enkel agent i et landskap kan formuleres som:

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer en agent, L(t) landskapet og N(t) er relasjoner mellom agent og landskap.

Nettverket har relasjonen N(t)={A(t), L(t)} der A(t) representerer en agent med stasjonær tilknytting mellom agent og landskap.

I dette tilfellet representerer L(x,y,z,t) et sett av parallelle landskap der hvert landskap kan endre seg i forhold til tid og sted. konstant. Landskapsmodellen L(x,y,z,t) kan f.eks inneholde et stasjonært terreng landskap L(terreng,x,y,z,t) og et strøm landskap L(strøm,x,y,z,t) der hver posisjon (x,y,z) har en strøm vektor L(strøm).

Aktiv partikkel agentmodell A(t)
Aktive partikkel agenter har et formål. Det kan f.eks være å følge en egenskap ved landskapet. Agentens formål bestemmer igjen partikkel agentens egenskaper. Agenter kan modelleres med egenskapene

$$A(t)=\{A(ark,t), A(dyn,t), A(eti,t), L(lær,t)\}$$

Der A(ark,t) representerer agentens arkitektur, A(dyn,t) agentens tilstandsdynamikk, A(eti,t) agentens etikk, og L(lær,t) agentens læringsstrategi.

Agent arkitektur A(ark,t)
En typisk agent arkitektur A(ark,t) for en aktiv partikkel agent er:

$$A(ark,t)=\{Sensor, Body, Motor\}$$

Der Sensor måler og oppdaterer tilstanden til landskapet L(z,x,y,t), Body representerer agentens form som en visuell agent. Motor er en tjenester som ivaretar agentens framdrift. Typiske tjenester for en partikkelagent vil være:

A(Sensor,t) = {Avleser agentens nære omgivelse}.
A(Body,t) = {Representerer partikkelens geometri og friksjon}
A(Motor,t) = {Ivaretar partikkelens framdrift}

Agent dynamikk A(dyn,t)
Agent dynamikk A(dyn,t) representerer agentens tilstandsdynamikk. En typisk dynamisk modell for en partikkelmodell er:

Partikkel akselrasjon vektor: a(part.,t) = a(part, t)+F(motor,t)/masse
Partikkel kraftvektor: F(motor,t) = -K(friksjon)v(rel,t)+F(ekstern,t)
Relativ partikkel hastighet: v(rel,t) = v(strøm)-v(part,t)
Partikkel hastighet vektor: v(part,t+T) =v(part,t)+T*a(part,t)
Partikkel posisjon vektor: x(part,t+T) = x(part,t)+T*v(part,t)

Der T er samplingsintervallet. K(friksjon) representerer friksjons koeffisienten mellom partikkelen og partikkelens relative hastighet v(rel).

Agent etikk A(eti,t)
Aktive partikkel agenter har et formål. Det vil si at den er tillagt en fri vilje til å realisere et mål som kan realiseres innenfor en ressurs eller en annen begrensning eller yteevne. Det kan formuleres som:

A(eti,t) = {Mål, Yteevne}

Der Mål representerer en ønsket tilstand for den aktive partikkel agenten. Yteevne representerer agentens ressurs eller kapasitet for å kunne nå målet. Ved simulering av aktive partikkelagenter er yteevnen knytter til agentens arkitektur, tilstand dynamikk og evne til læring.

Agent læring A(lær,t)
Agent læring er realisering av agentens etikk der læringsbegrepet er knyttet til egenskapene

A(lær, t) = {Identifikasjon, Kontroll}

Der Identifikasjon representerer en måling av landskapet i agentens omgivelser.

Kontroll = (Mål-Identifikasjon)K
Kontroll er altså hvordan en legger vekt på avviktet mellom mål og identifisert tilstand.

FRAMGANGSMÅTE

Agentbasert simulering av passiv partikkelmodell kan betraktes som et laboratorium eksperiment der en tar utgangspunkt i følgende framgangsmåte.

1. Sett opp en meta systemmodell for eksperimentet
2. Sett opp en agentmodell
3. Formuler agentens formål og metoder
4. Programmer agentens egenskaper
5. Dokumenter agentens egenskaper
6. Diskuter dine resultater

Framgangsmåte ved metodeutvikling:

1. Formulert fysiske egenskaper ved landskapet L(t)

2. Formuler partikkel agentens formål og mål i landskapet L(t)
3. Formuler en modell for agentens motor og egendynamikk
4. Formuler agentens læringsmetode for kontroll av egen tilstand

OPPGAVE

Kontrollspørsmål om partikkel agenter

1. Gjør rede for hva som menes med en aktiv partikkelagent.
2. Hva kan en aktiv partikkelagent representere?
3. Hvordan kan vi lage en fysikkmodell for en aktiv partikkelagent?

Forsøk med aktive partikkel agenter

I dette skal vi studere hvordan en gruppe aktive partikkel agenter, med en valgt masse, tilpasser seg et ønsket høydenivå i et lukket landskap.

1. Ta utgangspunkt i forrige oppgave og studer hvordan en gruppe aktive partikkel agenter oppfører seg i landskapet.
2. Gjør noen forsøk der agentene har tilfeldig masse og tilpasser seg en tilfeldig posisjon.

3.4 Produsent Agenter

Spørsmål: Hva er en Produsent Agent?
Svar: En Produsent Agent er en konsumerer ressurser, og som transformerer ressursen over på en ny form.

Spørsmål: Hva er eksempler på Produsent Agenter?
Svar: Typiske eksempler på Produsent Agenter, er agenter som skal representere industrielle eller biologiske systemer. Noen eksempler på industrielle systemer er biler, skip, roboter og produksjonssystemer. Noen biologiske eksempler er agenter som representerer planter, trær, dyr, personer og liknende.

Spørsmål: Hva kan Produsent Agenter gjøre av oppgaver?
Svar: Typiske oppgaver er løse optimaliseringsproblem. Det kan være å optimal ressursutnyttelse, produksjonskapasitet, lokal risiko, tidspunkt for optimal vekst, utbredelse av vegetasjon, utbredelse av marked osv.

Spørsmål: Hvordan kan en modellere Produsent Agenter?
Svar: Produsent agenter kan modelleres ved at den søker en ressurs fra et landskap eller fra andre agenter og så produserer en ny ressurs. Produsent agenter og biologiske agenter transformerer gjerne et materiale fra en form til et annet. Agenter som representerer biler, skip eller roboter, transformerer gjerne en ressurs om til en form for bevegelse. Agentens optimaliseringsproblem er da gjerne å tilpasse sin kostnad og produksjonskapasitet i forhold til ressurstilgang.

Spørsmål: Hvordan kan en agent velge en ressurs?
Svar: Produsent agenten kan velge en ressurs ved å sette en verdi på ressursen, og en kostnad på utførelsen av transformasjonen. Agenten velger da ressurser etter en kost/nytte-vurdering.

LÆRINGSMÅL

Studenten skal ved avslutningen av denne leksjonen kunne:

1. Gjøre rede for anvendeleser av Produsent Agenter
2. Modellere en Produsent Agent
3. Utføre et eksperiment med anvendelse av Produsent Agenter

SYSTEMTEIRI

Metasystemet for en enkel agent i et landskap kan formuleres som:

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer en agent, L(t) landskapet og N(t) er relasjoner mellom agent og landskap. Nettverket har relasjonen N(t)={A(t), L(t)} der A(t) representerer en agent med stasjonær tilknytting mellom agent og landskap.

Landskap L(t)

Tilstanden til konsument agenter har sammenheng med forholdet mellom en produksjonsprosess i landskapet L(t) og en konsument prosess i agenten A(t). Dette fører til en tilbakekobling prosess der

$$L(t+T) = f(A(t)+L(egen,t)) \text{ og}$$
$$A(t+T) = f(L(t))$$

per T er et samplingsintervall og L(egen,t) representerer en egenproduksjon i landskapet. En typisk vekstmodell kan modelleres som en logistisk modell. Denne kan formuleres som:

$$L(t+T)=(1+T*a)L(t)-(T*b)\ L(t)*L(t)$$

der a representerer vekstraten og b representerer reduksjonsraten i landskapet.

Agentmodell A(t)

Agenter kan modelleres med egenskapene

$$A(t)=\{A(ark,t),A(dyn,t), A(eti,t), L(lær,t)\}$$

der A(ark,t) representerer agentens arkitektur, A(dyn,t) agenten tilstandsdynamikk, A(eti,t) agentens etikk, og L(lær,t) agentens læringsstrategi.

Agent arkitektur A(ark,t)

En typisk agent arkitektur A(ark,t) for en stedbundet agent vil være:

$$A(ark,t)=\{Sensor, Body, Producer, Controller\}$$

der Sensor måler og oppdaterer tilstanden til landskapet L(z,x,y,t), Body representerer agentens form som en visuell agent. Producer transformerer konsumerte ressurser til en ny tilstand, eller en ny ressurs. Kontroller utøver en kontroll for læring av agentens etikk. I dette tilfellet kan en tenke seg at agenten har følgende tjenester:

$$A(Sensor,t)=\{\text{Avlese tilstand i landskapet}\}$$
$$A(Body,t)=\{\text{En enkel geometri}\}$$
$$A(Producer,t)=\{\text{En transformasjon fra et konsum til et produkt}\}$$
$$A(Controller,t)=\{\text{Kontroll av konsum}\}$$

Agent etikk A(eti,t)
Agentens etikk har egenskapene A(eti,t) = {mål, begrensninger}. Det vil si at agenten søker å realisere mål innenfor rammen av agentens begrensninger. Typiske egenskaper vil her være A(eti,t) = {optimal konsum over tid, kapasitet}.

I dette tilfellet kan en tenke seg at agentens etikk er:

A(eti,t)={Å optimalisere eget konsum i landskap over tid}

Agentens konsum i et landskap danner grunnlag for agentens produksjon, som igjen er forbundet med en kostnad.

Agent læring A(lær,t)
Agent læring er å utøve den kontoll som skal til for å realisere agentens etikk. Denne læringen har ett perspektiv knyttet til omgivelsene og ett knyttet til agenten. Dette kan sammenfattes med egenskapene A(lær,t) = {Identifikasjon, Kontroll}. Det vil si en identifikasjon av landskapets tilstand, og en kontroll av agentens tilstand. Landskapets tilstand identifiseres med agentens sensor, via agentens nettverk.

Læring kan utøves i samsvar med ulike typer kontroll strategier. En enkel metode er å utøve kontroll i samsvar med kontrolloven

$$U(t)=[R-L(t)]K(t) \qquad \text{for } U(t) > 0$$

der R er ønsket nivå på landskapet, L(t) er identifisert tilstand i landskapets tilstand L(x,y,t) og K er agentens vektlegging på avviket.

Optimal læring
Optimal læring er da å optimalisere kostnadsfunksjonen

J = (Konsum verdi– Kostnad)

Det vil si at agentens valg av ressurser i landskapet, må settes i sammenheng med den kostnad som er forbundet med å hente ressursen.

FRAMGANGSMÅTE

Anvendelsen av Transform Agenter kan betraktes som et laboratorium eksperiment der en tar utgangspunkt i følgende framgangsmåte.

1. Sett opp en meta systemmodell for eksperimentet
2. Sett opp en agentmodell
3. Formuler agentens formål og metoder
4. Programmer agentens egenskaper
5. Dokumenter agentens egenskaper
6. Diskuter dine resultater

Framgangsmåte ved metodeutvikling:

1. Formulert fysiske egenskaper ved landskapet L(t). Dersom landskapet produserer en ressurs, formulerer du en modell for egenutviklingen av landskapets dynamiske utvikling
2. Formuler ressursens verdi i landskapet
3. Formuler agentens kostnad ved å konsumere en ressurs i landskaper, og transformere ressursen over på en ny form.
4. Formuler agentens driftkostnad.

5. Formuler en strategi for hvor mye av ressursen agenten vil konsumere

OPPGAVE

Kontrollspørsmål om produksjon agenter

1. Gjør rede for hva som menes med Produksjon Agenter,
2. Hva kan Produksjons agenter representere?
3. Hvordan kan en Produksjon agent tilpasse egen produksjonskapasitet til ressurser i et landskap?

Forsøk med produksjon agenter

I dette skal vi studere hvordan en enkel Produksjons agent, kan tilpasse sin egen produksjon kapasitet til en ressurs som produseres i et landskap. Vi tenker oss da at landskapet produserer en ressurs i samsvar med en logistisk modell

$$L(t+1)=aL(t)-bL(t)L(t)$$

Der a og ber er valgte konstanter.

Produksjons Agenten velger en mengde fra landskapet L(t), så lenge mengden i L(t) > 0. Videre tenker vi oss at en mengdeenhet dL fra landskapet L(t) har en verdi Vo. Produksjon Agenten har en kostnad Ko som er forbundet med å produserer en tjeneste To fra verdien Vo.

Skriv en metode for der en Produksjon agent som konsumerer ressurser fra det dynamiske landskap L(t), og som produserer en mengde verditjenester Vo.

Hvordan kan Produksjons agenten lære å bli et stabilt produksjonssystem?
Hvordan kan Produksjon agenten optimalisere verdien av sin produksjon?

3.3 Målorientert bevegelige agenter

Spørsmål: Hva er målorienterte bevegelige agenter?
Svar: Det er agenter som har evne til å bevege seg til bestemte mål i et landskap.

Spørsmål: Hva er eksemplene på målorienterte bevegelige agenter?
Svar: Eksempler på målorienterte bevegelige agenter er agenter som representerer personer, biler, skip osv og som har en reiserute mellom to eller flere punkter.

Spørsmål: Hva kan målorienterte agenter gjøre?
Svar: I praktisk simulering av komplekse systemer er det slik at agentene skal representere personer, biler, skip, osv. Agentene må da kunne bevege seg from til fastsatte mål i mer eller mindre komplekse landskap. Agentens reiserute i komplekse landskap kan være kjent, den kan være ukjent, og den kan være beheftet med hindringer. Noen typiske oppgaver for agentene vil være å beregne reisetider, kostnader, reisemønster og risikofaktorer.

Spørsmål: Hvordan kan en vi få en agent til å nå mål i komplekse landskap?
Svar: Svaret her er avhengig av egenskapene til landskapet. Den enkleste metoden er å sette opp en reiserute via et sett med delmål. Så lar en agenten følge reiseruten fram til det endelige målet. Dersom reiseruten ikke er fastsatt via delmål, må agenten følge landskapet og prøve seg fram. Dersom der er flere mål å velge mellom, må agenten velge et mål. Dette valget kan gjøres på

grunnlag av en verdi, eller kostnadsbetraktning.

Spørsmål: Hvordan kan en vi få en agent til å nå ett av flere mål i komplekse landskap?
Svar: Den enkleste metoden er å sette verdier på mål og så beregne kostnader forbundet med å nå mål. Agenten velger så det mål som gir størst verdiskaping.

LÆRINGSMÅL

En skal ved avslutning av denne leksjonen

Kunnskaper

1. Kunne lage en systemmodell for målorienterte agenter
2. Kunne modellere egendynamikken til målorienterte agenter
3. Kunne innføre kontroll strategier for målorienterte agenter

Ferdigheter

1. Kunne utnytte Unity spillmotor til simulering av målorienterte agenter
2. Kunne programmere målorienterte
3. Kunne velge kontroll strategier for kontroll av molorienterte agenter

Generelle kunnskaper

1. Ha kunnskaper om bruksområder til målorienterte agenter
2. Ha kunnskaper om muligheter og begrensninger ved målorienterte agenter

TEMA

1. System modellering av målorienterte agenter
2. System arkitektur for målorienterte agenter
3. Verdisetting av målvektorer
4. Læring av målorienterte agenter
5. Programmering av bevegelige agenter
6. Valgstrategier

SYSTEMTEORI

Metasystem

Metasystemet S(t) for bevegelige agenter kan fortsatt formuleres som:

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer en agent, L(t) er landskapet og N(t) er relasjoner mellom agenter og landskap. I dette tilfelle har agenten A(t) en posisjon (x1,y1,z1) i landskapet L(t) før den forflyttes til en posisjon (x2, y2, z2).

Agentmodell

Agentens interne egenskaper kan modelleres med system perspektivene

$$S(t)=\{A(arc,t), A(dyn,t), A(eti,t), A(lea,t)\}$$

der A(arc,t) representerer agentens arkitektur, A(dyn,t) agentens dynamikk, A(eti,t) agenten etikk og A(lea,t) agentens læring.

Agent arkitektur A(arc, t)

Bevegelige agenter kan modelleres med følgende tjenester:

A(arc,t)={Sensor, Body, Motor, Controller, Output}

Der Sensor måler agentens posisjon I landskapet, Body representerer agentens egendynamiske egenskaper, Motor gir agenten framdrift og Controller utøver kontroll av agentens hastighet og retning. Hver tjeneste kan ha et sett av spesifikasjoner.

A(Sensor,t)={Agentens målinger til landskap eller andre agenter}
A(Body,t)={Visuell modell, volum, vekt, friksjon,++}
A(Motor,t)={Kraft pådrag, ror,++}
A(Controller,t)={Controll av hastighet, retning, start, stop,++}
A(Output,t)={Pådrag til landskap eller andre agenter}

Agent dynamikk A(dyn, t)
Agenter har en egendynamikk som normalt fastsetter i samsvar med balansemodeller. Noen typiske dynamiske modeller for en bevegelig agent vil være vektorene:

Kraft:	F(motor,t)	Kraftpådrag til dynamisk modell
Akselrasjon:	a(t)=F(t)/m	(kraft/masse)
Friksjon:	F(frik,t)=-Kf*v(t)	(kraft)
Oppdrift:	F(opp,t)= (m-ro*V)g	(kraft)
Hastighet:	v(t+T)=v(t)+a(t)	(meter/sek)
Posisjon:	x(t+T)=x(t)+v(t)	(z,x,y)

Agent etikk A(eti, t)
Agenters etikk har egenskapene A(eti,t) = {mål, begrensninger}. Det mål representerer en målet for agentens tjeneste og begrensninger representerer egenskaper ved agentens utforming eller parametre.

Målorienterte agenter skal holde riktig hastighet samtidig som de skal nå riktig mål. De har derfor en mer komplisert målstruktur enn bare bevegelige agenter. En typisk målstruktur for en bevegelig agent er:

1. Tilstandsmål = (Hastighet, retning,++)
2. Posisjonsmål =(Målposisjon, Type, Verdi)
3. Oppførsel = (Posisjonsmål, Tilstandsmål)

Posisjonsmål:
Agentens posisjonsmål har Målposisjon som definerer en (x,y,z) posisjon il landskapet. Type er en klassifikasjon av hva målet representerer. Verdi er en klassifikasjon av målets ressurs egenskaper. Posisjonsmålene samles i en vektor.

Tilstandsmål:
Her er tilstandsmål er mål som er knyttet mål for agentens egendynamikk. Typiske tilstandsmål et mål for agentens hastighet og retning. Tilstandsmålene samles i en vektor.

Oppførsel:
Begrepet oppførsel er knyttet endringer i agentens aktiviteter. Agentens aktiviteter er igjen knyttet til agentens målvektor. Agentens samlede oppførsel er et sett av posisjonsmål og tilstandsmål.

Agent læring A(lær, t)

Agentens læring A(lær, t) er å realisere agenten etikk A(eti,t). Læring av agentens målstruktur er i dette tilfellet:

1. Tilstand læring: Læring av posisjon og hastighet
2. Posisjon læring: Læring å nå riktig posisjon
3. Oppførsel læring: Læring av rekkefølgen på delmål

Det nye her er altså innføring av begrepet oppførsel, der en kontrollfunksjon utøver kontroll av agentens tilstander.

Bevegelse mott ett mål:
En enkel metode for å regulere agentens posisjon opp til et mål er modellen:

$$Vtx(t)=log(Gtx*|Etx(t)|+1)$$

Der Vtx(t) er ønsket hastighet fram til måtet, Gtx er en forsterkning og |Etx(t)| er agentens avstand fram til målet.

Valg av mål:
Dersom agenten må velge ett blant flere mål, må agenten ha et kriterium for valg av mål. En metode for å velge mål er å velge det mål som gir maksimal verdi til agenten. Dette kan gjøre med regelen

Posisjonsmål =Maks((Målposisjon, Verdi) – Målkostnad))

Der Målkostnad er agentens beregnede kostnad for å nå en posisjon.

Læring av oppførsel
Begrepet oppførsel er knyttet til agentens tilstand mellom to eller flere mål i en målstruktur. Oppførsel kontroll er å kontroller agentens tilstander i målstrukturen. Dette kan gjøres på flere måter. Dersom reiseruten er kjent, kan vi lage en målstruktur med reiserute, som styrer en tilstandsmaskin, som igjen styrer agentens tilstander.

FRAMGANGSMÅTE

Tilfelle 1: Ett stasjonært mål

System modellering

1. Sett opp en systemmodell for agentens arkitektur A(ark,t)
2. Formuler agenten etikk og sett mål for agentens å nå en posisjon
3. Sett opp agentens målvektorer
4. Sett opp en strategi for kontroll av agentens bevegelse mot målet
5. Sett opp kriterier for vurdering av agenten yteevne

Programmering

1. Ta utgangspunkt i programmet for Bevegelig Agent
2. Innfør en målvektor i tjenesten A(Body)
3. Innfør kontroll av agentens posisjon i tjenesten i A(Controller)
4. Sett inn en kostnadsfunksjon i agentens A(Body) som overvåker agentens avvik
5. La A(Output) rapportere agentens hastighet, retning og kostnader

Vurderinger

1. Sett opp en testplan for hvordan du vil teste og vurdere agentens yteevne.

TILFELLE 2: N-DELMÅL
System modellering

1. Sett opp en systemmodell for agentens arkitektur A(ark,t)
2. Formuler agentens etikk og sett mål for å nå en posisjon
3. Sett opp agentens målvektorer med mål, type og verdier
4. Sett opp en strategi for valg av mål med maksimum verdi
5. Sett opp kriterier for vurdering av agenten yteevne

Programmering

1. Ta utgangspunkt i programmet for Bevegelig Agent
2. Innfør en målvektor i tjenesten A(Body).
3. Innfør kontroll av agentens posisjon i tjenesten i A(Controller)
4. Sett inn en kostnadsfunksjon i agentens A(Body) som overvåker agentens avvik.
5. La A(Output) rapportere agentens hastighet, retning og kostnader

Vurderinger

1. Sett opp en testplan for hvordan du vil teste og vurdere agentens yteevne.

TILFELLE 3: BEVEGELIGE MÅL
System modellering

1. Sett opp en systemmodell for agentens arkitektur A(ark,t)
2. Formuler agentens etikk og sett mål for å nå et bevegelig mål
3. Sett opp målvektorer med mål, type og verdier til det bevegelige målet
4. Sett opp kriterier for vurdering av agenten yteevne

Programmering

1. Ta utgangspunkt i programmet for Bevegelig Agent
2. Innfør en målvektor i tjenesten A(Body)
3. Innføre en tjeneste i A(Sensor) som oppdaterer posisjonen til det bevegelige målet
4. La A(Controller) styre agentens hastighet og retning mot det bevegelige målet.
5. La kostnadsfunksjon overvåker agentens yteevne.
6. La A(Output) rapportere agentens hastighet, retning og kostnader

OPPGAVE

Kontrollspørsmål om bevegelig målorienterte agenter

1. Gjør rede for hva som menes med en bevegelig målorienterte Agenter
2. Hva de kan bevegelig målorienterte agenter representere?
3. Gjør rede for hvordan bevegelig målorienterte agenter kan styre sin hastighet og retning mot et mål.
4. Gjør rede for hvordan bevegelig målorienterte agenter kan styre sin hastighet og retning mot et mål.
5. Gjør rede for hvordan bevegelig målorienterte agenter kan velge ett av flere mål.

Forsøk bevegelig målorienterte agenter

I denne oppgaven skal vi utvikle bevegelig målorienterte agent.

1. System modellering: Opprett en systemmodell S(t) for målorientert agent.
2. Forsøk med ett mål: Ta utgangspunkt i Bevegelig Agent med treghet og

programmer en tjeneste slik at den kan bevege seg med konstant hastighet fram til et stasjonært mål. Studer agentens yteevne ved å måle evne til posisjonere seg til målet.

3. Forsøk med et bevegelig mål: Lag en ny metode som gir agenten evne til å nå et bevegelig må, som går i sirkel. Studer agentens yteevne ved å måle evne til posisjonere seg til målet.
4. Forsøk med flere mål: Sett opp en et sett med mål, med tilfeldige posisjoner og verdier. Sett opp en regel for valg av mål med maksimum verdi og studer agentens evne til å optimalisere tilgang til verdier.
5. Kommenter dine resultater.

3.4 Intelligente agenter

Spørsmål: Hva er en intelligent agent?
Svar: En intelligent agent er en agent med evne til å overvåke og optimalisere sin egen yteevne.

Spørsmål: Hvordan kan agenten overvåke og optimalisere egen yteevne?
Svar: En vanlig metode er å sammenlikne yteevnen fra kostnadsfunksjoner. Prinsippet her er at agenten velger den parameter som gir minst kostnader fra kostnadsfunksjonen. Dette kan sammenliknes med en regulator som josterer en parameter til minst avvik.

Spørsmål: Hva er typiske anvendelser på intelligente agenter?
Svar: Intelligente agenter er nyttige når agentens parametre er ukjent. En kan da overlate ansvaret til agenten om å tilpasse sine egne parametre. Et annet eksempel er når agenten optimalisere egne parametre til endringer i omgivelsene. Et typisk eksempel her er kontroll av hastighet på skip med endring i last eller bunnforhold.

LÆRINGSMÅL
En skal ved avslutning av denne leksjonen
Kunnskaper

1. Kunne gjøre rede for hva en intelligent agent er for noe
2. Kunne gjøre rede for hva intelligente agenter skal kunne benyttes til

Ferdigheter

1. Kunne utnytte Unity spillmotor til simulering av bevegelige agenter
2. Kunne programmere intelligente agenter
3. Kunne velge kontroll strategier for intelligente agenter

Generelle kunnskaper

1. Ha kunnskaper om bruksområder til intelligente agenter
2. Ha kunnskaper om intelligente agenters muligheter og begrensninger

TEMA

1. System modellering av intelligente agenter
2. System arkitektur for intelligente agenter
3. Intelligente agents egendynamikk
4. Kostnadsfunksjoner
5. Læring av intelligente agenter

SYSTEMTEORI

Metasystemet S(t) for intelligente agenter kan fortsatt formuleres som:

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer en agent, L(t) er landskapet og N(t) er relasjoner mellom agenter og landskap. I dette tilfelle har agenten A(t) en posisjon (x1,y1,z1) i landskapet L(t) før den forflyttes til en posisjon (x2, y2, z2). Agentens interne egenskaper kan modelleres med system perspektivene

$$S(t)=\{A(arc,t), A(dyn,t), A(eti,t), A(lea,t)\}$$

der A(arc,t) representerer agentens arkitektur, A(dyn,t) agentens dynamikk, A(eti,t) agenten etikk og A(lea,t) agentens læring.

Agent arkitektur A(arc, t)

Intelligente agenter har i prinsippet samme arkitektur som andre bevegelige agenter med kan egenskapene

$$A(arc,t)=\{Sensor, Body, Motor, Controller, Output\}$$

Der Sensor måler agentens posisjon i landskapet, Body representerer agentens egendynamiske egenskaper, Motor gir agenten framdrift og Controller utøver kontroll av agentens hastighet og retning.

Agent dynamikk A(dyn, t)

Intelligente agenter har i prinsippet samme egendynamikk som andre bevegelige målorienterte agenter. Forskjellen ligger i at de har større evne til i å optimalisere hastighet og retning ut fra vektfunksjoner eller kostnader.

Agent etikk A(eti, t)

Agenters etikk har egenskapene A(eti,t) = {mål, begrensninger}. Det vil si at de skal nå egne mål og samtidig justere egne optimalisere egne begrensninger. Det er da slik at mål og identifikasjon av begrensninger, er to sider av samme sak. Når en intelligent skal nå et mål, kan agentens etikk formuleres som

$$A(eti,t) = \{Variabelt\ mål, variabel\ begrensning\}.$$

Det blir at optimaliseringskriteriet som bestemmer agentens valg av mål og begrensninger eller egenskaper.

Agent læring A(lea, t)

Agent læring A(lær, t) er å realisere agenten etikk A(eti,t). Læring av intelligente agenter vil si å til passe agentens yteevne eller parametre. En typisk modell for kontroll av hastighet er:

$$F(motor, t) = (R(hast, t)-v(hast,t))K(hast,t) = e(t)K(hast,t)$$

Der F(motor, t) er kraft pådrag til motor, R(hast, t) er ønsket hastighet, v(hast, t) er virkelig hastighet og e(t) hastighet avviket. K(hast,t) er parameter for hastighetskontroll som bestemmer agenten yteevne.

Overvåke yteevne
Agentens yteevne kan beregnes via en kostnadsfunksjon. Et eksempel på en kostnadsfunksjon er:

$$J(t) = e(t)Q+u(t)R$$

Der e(t) representerer avvik fra mål, Q er en valgt vektlegging eller kostnad på avviket, u(t) er pådrag (kraft eller energi) på tjenester, R er valg vektlegging eller kostnad på pådraget. Er der usikkerhet i målingen av e(t) eller u(t), kan det være fornuftig å estimere tilstanden til kostnadsfunksjonen J(t) til:

$$J(t+T) = La*J(t)+(1-La)[e(t)Q+u(t)R]$$

Der T er et samplingsintervall, La er en valgt konstant La <1, som bestemmer hvor raskt J(t) skal nå en middelverdi.

Tilpasse yteevne
Intelligente agenter overvåker sin yteevne for så å justere sin yteevne. I dette tilfellet er yteevnen knyttet til avvik fra ønsket hastighet og retning. Der finnes en omfattende litteratur om hvordan en kan innføre en selvjusterende tilpassing av agentens yteevne. Vi skal her bare antyde noen enkle metoder:

Adaptiv kontroll
Noen enkle eksempler på adaptiv kontroll er:

$$K(hast,t) = Ko*J(t) \quad \text{og}$$
$$K(hast,t) = Ko*J(t)/(R(hast)-J(t))$$

Der Ko er en valgt konstant og R(hast) er ønsket hastighetsvektor.

Kontroll via Neuralt nettverk
En annen metode er å benytte et neuralt nettverk som en selvregulerende forsterkningsfaktor K(hast,t). Et neuralt nettverk har egenskapene

$$Nout = f(NN, Targt, Input)$$

Der NN representerer nettverkets egenskaper, Target er tilstanden som skal identifiseres, Input er pådraget til nettverket og Nout er responsen fra nettverket. En kontrollstrategi, basert på et neuralt nett kan da f.eks implementeres ved å sette

$$K(hast,t) = f(NN, Targt=R(hast), Input=v(hast,t))$$

Nettverket NN må så trenes opp kontinuerlig.

Kontroll via Genetisk algoritme
Innføring av en selvjusterende kontroll kan også gjøres med en genetisk algoritme. Dette kan gjøres ved å ta utgangspunkt i en PI-regulator. En enkel kontroll strategi K(hast,t), basert på en PI-regulator, kan formuleres som.

$$K(hast,t) = K0 + 1/x(t+1), \text{ for } x(t+1)=ax(t)+e(t)$$

Der Ko representer forsterkningen i proporsjonal leddet, og a representerer integrasjonsfaktoren i integrasjonsleddet. I en genetisk algoritme er Ko og a frie variable som gener og optimalisert i forhold til kostnadsfunksjonen J(t) og avvik e(t).

FRAMGANGSMÅTE

System modellering

1. Sett opp en systemmodell for agentens arkitektur A(ark,t)
2. Formuler agenten etikk og sett mål for agentens hastighet og retning
3. Sett opp balansemodeller for egendynamikk med masse, friksjon og forstyrrelse
4. Sett opp en strategi for kontroll av agentens hastighet og retning
5. Sett opp en strategi for kontroll og vurdering av agenten yteevne

Programmering

Ta utgangspunkt i resultatene fra arbeidet med målorienterte bevegelige agenter

OPPGAVE

Kontrollspørsmål om intelligente agenter

1. Hva er en intelligent agent?
2. Hva kan intelligente agenter benyttes til?
3. Hvordan kan intelligente agenter modelleres?

Forsøk med intelligente agenter

En intelligent agent skal representere et bevegelig objekt som forflytter seg i et landskap. Agenten har en masse M og en friksjonskoeffisient Kf. Ønsket hastighet til den bevegelige agenten, reguleres av en PID-regulator.

1. Ta utgangspunkt i tidligere oppgave om målorienterte bevegelige agenter.
2. Sett opp en modell for agentens egenskaper med masse, friksjon og forstyrrelse
3. Formuler et eksperiment der den intelligent agenten har evne til å identifisere egne parametre til PID-regulatoren.
4. Velg en metode for overvåking av agentens yteevne
5. Kommenter resultatet fra forsøkene

4 Intelligente systemer

Spørsmål: Hva er et intelligent objekt?
Svar: Et intelligent objekt er en objekt med evne til å overvåke og optimalisere sin egen yteevne.

Spørsmål: Hva er et intelligent system?
Svar: Et intelligent system harr en når tre eller flere objekter overvåker og optimaliserer sin egen yteevne.

Spørsmål: Hva menes med begrepet smart objekt?
Svar: Et smart objekt er det samme som et intelligent objekt. Dette er begreper som brukes om hverandre, men er i realiteten det samme.

Spørsmål: Hva menes med begrepet smart system?
Svar: Et smart system er det samme som et intelligent system.

Spørsmål: Hva menes med begrepet smart grid?
Svar: Et smart grid er et nettverk som overvåker og optimaliserer sine egne tjenester.

Spørsmål: Hvordan kan agenten overvåke og optimalisere egen yteevne?
Svar: En vanlig metode er å sammenlikne yteevnen fra kostnadsfunksjoner. Prinsippet her er at agenten velger den parameter som gir minst kostnader fra kostnadsfunksjonen. Dette kan sammenliknes med en regulator som josterer en parameter til minst avvik.

Spørsmål: Hva er typiske anvendelser på intelligente agenter?
Svar: Intelligente agenter er nyttige når agentens parametre er ukjent. En kan da overlate ansvaret til agenten om å tilpasse sine egne parametre. Et annet eksempel er når agenten optimalisere egne parametre til endringer i omgivelsene. Et typisk eksempel her er kontroll av hastighet på skip med endring i last eller bunnforhold.

LÆRINGSMÅL
En skal ved avslutning av denne leksjonen
Kunnskaper
1. Kunne gjøre rede for hva begrepet intelligente systemer
2. Kunne identifisere typiske egenskaper ved intelligente systemer

Generelle kunnskaper
1. Ha kunnskaper om bruksområder til intelligente systemer
2. Ha kunnskaper om intelligente ystemers muligheter og begrensninger

TEMA
1. Intelligente objekter og systemer
2. Smart grid
3. Smarte tjenester
4. Smarte byer

OPPGAVE
Kontrollspørsmål om intelligente systemer
1. Gjør kort rede for hva som menes med begrepene intelligente objekter og intelligente systemer
2. Gjør kort rede for hva som menes med begrepet Smart Grid
3. Hva er karakteristiske egenskaper ved Smarte hus?
4. Hva er karakteristiske egenskaper ved Smart energiforvaltning?
5. Hva er karakteristiske egenskaper ved Smarte byer?

5 Sosiale Agenter

Spørsmål: Hva er sosiale agenter?
Svar: Sosiale agenter er et sett agenter som samarbeider med samme regler innenfor et felles formål.

Spørsmål: Hva er eksemplene på sosiale agenter?
Svar: Eksempler på sosiale agenter er agenter som representerer en gruppe med personer, fisk, biler, skip eller partikler som opptrer innenfor et felles sett med regler.

Spørsmål: Hva kan sosiale agenten gjøre for deg?
Svar: Sosiale agenter kan løse optimaliseringsproblem. De kan for eksempel simulere optimale produksjonssystemer, logistikk, identifisere optimal kapasitet i køer, oppførsel til trafikkmodeller og individbaserte markedsmodeller.

Spørsmål: Hvordan kan jeg modellere sosiale agenter?
Svar: Sosiale agenter kan modelleres som en flokk agenter med et felles sett med regler. Hver agent i flokken tilpasser da sin egen aktivitet i forhold til alle de andre, uten noen form for sentral kontroll.

Demo videoer

1. Swarm intelligence lecture: http://www.youtube.com/watch?v=IqhI4tWkyFc
2. Swarm lecture: http://www.youtube.com/watch?v=-G66iL__VdA

LÆRINGSMÅL

Kunnskapsmål

En skal ved avslutningen av denne leksjonen kunne:

1. Gjøre rede for metoder for systemmodellering av sosiale agenter
2. Gjøre rede for hvordan sosiale agenter kan benyttes til å løse oppgaver

Ferdighetsmål

En skal ved avslutningen av denne leksjonen kunne:

1. Sette opp et eksperiment ved bruk av sosiale agenter som problemløsere
2. Programmere grupper med sosiale agenter
3. Innføre kontrollstrategier for sosiale agenter

Generelle kunnskapsmål

En skal ved avslutningen av denne leksjonen:

1. Ha kunnskaper om anvendelse av sosiale agenters
2. Sette opp et eksperiment med sosiale agenter

EMNER

1. Agenter som grupper og flokker
2. Sosiale agenters arkitektur
3. Cohesion mellom sosiale agenter
4. Separation mellom sosiale agenter
5. Aligmnent for sosiale agenter
6. Sosiale agenter i køer
7. Optimalisering av agentegenskaper i køer

SYSTEMTEORI

Sosiale agenter er en del av metasystem

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett med agenter, L(t) et sett med landskap og N(t) et sett med relasjoner mellom agenter i et landskap. I dette tilfellet er relasjonene N(t) bestemt av agenten posisjon i forhold til de andre agentene. Agentens interne egenskaper kan modelleres med system perspektivene

S(t)={A(arc,t), A(dyn,t), A(eti,t), A(lea,t)}

der A(arc,t) representerer agentens arkitektur, A(dyn,t) agentens dynamikk, A(eti,t) agenten etikk og A(lea,t) agentens læring.

Sosiale agent etikk
En gruppe agenter A(t) er sosiale agenter når felles sosial etikk

A(eti,t)={Regler}

Der Regler representerer et felles med regler i agent gruppen A(t).

Sosial agent arkitektur
Sosial agent arkitektur sier noe om hvordan agentene er forbundet til hverandre via et felles nettverk. Dette kan formuleres som

S(ark,t)={N(t),A(t)}

Der N(t) representeres nettverket.

Sosial agent dynamikk
Sosial agent dynamikk er et begrep som sier noe om hvordan hele systemet S(t) utvikler seg over tid.

Sosial agent læring
Sosial agent læring er metoder for å nå egne mål via et felles sett med regler.

5.1 Agent svermer

Spørsmål: Hva er en agent sverm?
Svar: En agent sverm er sosiale agenter med en oppførsel som tilsvarer oppførselen til en sverm med fugler fisk eller liknende. Et karakteristisk trekk ved agent svermer er at de har ingen annen oppgave enn å holde gruppen sammen.

Spørsmål: Hva er eksemplene på agent svermer?
Svar: Eksempler på agent svermer er simuleringen av grupper med fisk, personer eller partikler som har en samlet flytbevegelse i et landskap.

Spørsmål: Hva kan agenten svermer gjøre for deg?
Svar: En agent sverm kan vise den samlede oppførsel og struktur for en gruppe agenter. Normalt er det slik at agenter *beiter* på en ressurs eller flykter fra en kostnad. Videre er det slik at en sverm er en basismodell for å kunne studere andre komplekse fenomener som kø fram til ressurser, markedsmodeller, trafikkmodeller, risikomodeller osv.

Spørsmål: Hvordan kan jeg modellere agenter svermer?
Svar: Modellering av agent svermer er baserest på tre prinsipper. Det ene prinsippet er at alle agenter i en gruppe har felles regler. Det andre prinsippet er at alle agentene har tilgang til tilstandene i de andre agentene. Det tredje prinsippet er at der er ingen overordnet kontroll. Alle agentene opererer uavhengig av de andre, men innenfor de samme regler.

LÆRINGSMÅL

Kunnskapsmål

En skal ved avslutningen av denne leksjonen kunne:

1. Gjøre rede for metoder for modellering av sosiale agenter
2. Gjøre rede for hvordan sosiale agenter kan benyttes til å løse oppgaver

Ferdighetsmål

En skal ved avslutningen av denne leksjonen kunne:

1. Sette opp et eksperiment ved bruk av sosiale agenter som problemløsere
2. Programmere grupper med sosiale agenter
3. Innføre kontrollstrategier for som samler sosiale agenter i en stabil struktur
4. Bevege sosiale agenter i en sirkel

Generelle kunnskapsmål

En skal ved avslutningen av denne leksjonen:

1. Ha kunnskaper om anvendelse av sosiale agenters
2. Sette opp et eksperiment med sosiale agenter

LÆRINGSMÅL

1. Systemteori for sosial agent
2. Sosiale agenter med Cohesion, Separation og Alignment
3. Horisontal læring av Cohesion, Separation og Alignment.

SYSTEMTEORI

I dette tilfelle har vi et metasystem

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett med agenter, L(t) landskapet og N(t) er relasjoner mellom agenter i landskap. I dette tilfellet er relasjonene N(t) bestemt av agenten posisjon i forhold til de andre agentene. Agenters etikk A(eti, t) er i dette tilfelle bestemt av hvilke felles sett regler agentene overholde innenfor gruppen A(t) med agenter.

Agentmodell

Agentens interne egenskaper kan modelleres med system perspektivene

$$S(t)=\{A(arc,t), A(dyn,t), A(eti,t), A(lea,t)\}$$

der A(arc,t) representerer agentens arkitektur, A(dyn,t) agentens dynamikk, A(eti,t) agenten etikk og A(lea,t) agentens læring.

Agent arkitektur A(arc, t)

Bevegelige agenter kan modelleres med følgende tjenester:

$$A(arc,t)=\{Sensor, Body, Motor, Controller, Output\}$$

Der Sensor måler agentens posisjon I landskapet, Body representerer agentens egendynamiske egenskaper, Motor gir agenten framdrift og Controller utøver kontroll av agentens hastighet og retning.

Agent dynamikk A(dyn, t)
Agenter i en sver er normalt målorienterte bevegelige agenter.

Agenters sosial etikk A(eti, t)
Sosiale agenter er basert på prinsippet om å overholde et felles sett ved regler. Hver agent i svermen reagerer i samsvar med noen enkle regler, uten noen som helst sentral kontroll. Resultatet av dette er at hele svermen oppnår en kompleks oppførsel. Agentens etikk har et eksternt og et internt perspektiv.

Ekstern etikk
Agentens eksterne etikk er knyttet til gruppen eller svermens felles formål. Et felles formål kan igjen være knyttet opp til et felles regelverk. Et typiske eksempel på et felles rekeverk er:

A(ekstern, eti)={Cohesion, Separation, Alignment}

Der Cohesion representerer en regel for samordning av agenter, Separation representerer en regel for separasjon av agenter og Alignment representerer en regel for retningsorientering av agenter.

Intern etikk
Agentens interne etikk er knyttet til agentens egne mål. Noen eksempler på agentens egne mål er:

A(Interne, eti)={Posisjon, Kostnader, Parametre,,,}

Agenters sosiale læring A(lear, t)
Agenters sosiale læring er tilpasse agentens tilstand i forhold til felles sosiale regler.

Cohesion
Læringsmetoden for Cohesion er basert på at agenten først beregner svermens virtuelle sentrum. Dette beregner hver agent beregner sin middelverdi i forhold til alle andre agenter. Cohesion sentrum og så middelverdien av dette. Dette kan formuleres som

Coh(pos) = Sum(A(j,pos)-A(i,pos))/(Antall-1)

Der A(i,pos) er agent i sin posisjon, som utfører en beregning. A(j,pos) er posisjonene til alle andre agenter i svermen. Hver agent har en dragning fra sin egen posisjon og mot et felles sentrum. Avstanden til det virtuelle punktet, for alle agentene, er

Coh(dist) = Coh(pos)-A(i,pos)

Der Coh(dist) er en avstandvektor. Retningen til svermens virtuelle midtpunkt er:

Coh(dir) = ||Coh(pos)-A(i,pos)||

Der || || representerer en retningsvektor for avstanden. Cohesion kraft er en egenskap ved agenten. Den sier noe om hvor mye vekt agenten legger på å ha en avstand fra det virtuelle punktet. Denne kraften kan beregnes med

F(coh)=Coh(dir)*Gcoh

der Gcoh representerer agentens kontroll strategi

Separation
Læringsmetoden for Cohesion er basert på at agenten beregner en motkraft som er omvendt proposjonel med avstanden til de nærmeste agentene. Dette kan gjøres med mange forskjellige metoden. En typisk metode er å beregne først relativ avstand

relDist = A(j,pos)-A(i,pos)

og så dele på kvadratet av avstanden til

relDist = relDis/[(A(j,pos)-A(i,pos))(A(j,pos)-A(i,pos))]

De nærmeste agentene vil da gi et størst bidrag til en relativ distanse relDist.
Separasjon avstanden blir da

Sum(relDist)

Eller summen av bidragene fra de nærmeste agentene. På den måten vil alle de nærmeste agentene bidra mer eller mindre med en separasjon kraft.

Alignment
Læringsmetoden for Alignment er basert på at agenten beregner den midlere sverm hastighet Svarm(velocity) for alle endre agenter i svermen. Dette kan formuleres som

Svarm(velocity) = Sum(A(j,velocity))/(Antall-1)

Der A(j,velocity) er hastigheten til en agent A(j,t). Agentens avvik, i forhold til svermens hastighet blir da

Align(velocity)=(Svarm(velocity)-A(i,velocity))

Feilavviket Alignment må så korrigeres in agentens framdrift.

FRAMGANGSMÅTE
Framgangsmåte for metode

1. Formuler først det eksperimentet agentene skal utføre for deg
2. Formuler så hvordan du vil teste resultatet av forsøket
3. Formuler så metodene du vil bruke.
4. Om du bruker Cohesion, Separation, Alignment må du tenke igjennom hvilken kontrollstrategi du vil brukef

Framgangsmåte for programmering
Når du skal beregne Cohesion, Separation og Alignment i Unity, må hver agent ha tilgang til posisjon og hastighet til alle andre agentene. Det kan gjøres via et egent objekt, som i programlistingen er kalt controller. Via controller har hver agent tilgang til listen med alle agentene. Hver agent går da inn i listen, trekker ut en og en agent, og avleser agentens tilstand. Se vedlagte kodeeksempel UpdateSwarState(). Hver agent kan så kontrollere sin egen tilstand, i forhold til de andre, med egne metoder.

OBLIGATORISK OPPGAVE

Kontrollspørsmål om sosiale agenter

1. Gjør kort rede for hva som menes med begrepet Sosiale agenter
2. Gjør kort rede for hva som menes med begrepet agent svermer
3. Gjør rede for Craig Reynolds sosiale regler for Cohesion, Separation og Alignment.

Forsøk med sosiale agenter

Det skal gjøres et eksperiment med sosiale agenter følger et mål som går i en sirkelbane.

1. Ta utgangspunkt i en tidligere bevegelig målagent som mål, og la den bevege seg i en sirkel.
2. Ta utgangspunkt i tidligere målorienterte agenter og opprett en sverm med agenter i et landskap.
3. Opprett en metode for Cohesion og studer hvordan agentene beveger seg mot et felles punkt. Juster kraftpådraget til agentene får en naturlig hastighet på tilnærmingen
4. Opprett en metode for Separation og studer hvordan agentene nå stiller seg inn i forhold til hverandre. Juster separasjon parameter til agentene får en passe avstand.
5. Opprett en metode for Alingment og studer hvordan agentene retning nå stiller seg inn i forhold til hverandre. Juster alignment parameter til agentene får en passe kontroll på retningen.
6. Kommenter resultatet du har kommet fram til

5.2 Agent svermkø

Spørsmål: Hva er en agent svermkø?
Svar: En agent svermkø er et sett sosiale agenter som konkurrerer om tilgang til en ressurs.

Spørsmål: Hva er eksemplene på agent svermkø?
Svar: Eksempler på agent svermer køer er simulering av trafikkmodeller der agentene skal passere en hindring som begrenser agentenes hastighet i flytretning. Andre eksempler er agenter som konkurrerer om tilgang til et felles mål.

Spørsmål: Hva kan simulering av agenten sverm køer gjøre for deg?
Svar: Simulering av sverm køer kan benyttes til å løse optimaliseringsproblem. Det kan være å identifisere kapasitet til en hindring som danner en kø, identifisere forventet antall i en kø, eller det kan være å identifisere strategier for å passere gjennom en kø.

Spørsmål: Hvordan kan jeg modellere agenter svermer køer?
Svar: Sverm køer kan modelleres som sosiale agenter. De har felles regler med Cohesion, Separation og Alignment som samordner køen. I tillegg søker hver agent et mål. Dette mål kan være felles for alle agentene.

LÆRINGSMÅL

Kunnskapsmål

En skal ved avslutningen av denne leksjonen kunne:

1. Gjøre rede for metoder for modellering av køer med sosiale agenter

2. Gjøre rede for hvordan sosiale agenter som køer kan løse oppgaver

Ferdighetsmål
En skal ved avslutningen av denne leksjonen kunne:
1. Sette opp et eksperiment bruk av sosiale agenter i køer
2. Programmere en kø med sosiale agenter
3. Innføre strategier for læring av agenter å tilpasse ser en kø situasjon
4. Innføre strategier for optimalisering av agenter i en køsituasjon

Generelle kunnskapsmål
En skal ved avslutningen av denne leksjonen:
1. Ha kunnskaper om anvendelse av sosiale agenters
2. Sette opp et eksperiment med sosiale agenter

SYSTEMTEORI

Metasystemet S(t)

I dette tilfelle har vi et metasystem

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett med agenter, L(t) landskapet og N(t) er relasjoner mellom agenter i landskap. I dette tilfellet er relasjonene N(t) bestemt av agenten posisjon i forhold til de andre agentene.

Agent etikk

Agenters etikk i en kø kan formuleres som:

$$A(eti, t) =\{Cohesion, Separation, Alignment, Mål\}$$

Vi ser her at agenter i en kø må forholde seg til de sosiale reglene Cohesion, Separation og Alignment. I tillegg er det nå kommet inn agentens Mål som en ny regel. Mål kan da være flere ting. Det kan være ønsket posisjon, ønsket verdi eller noe agenten ønsker å optimalisere.

Agent læring

Agenter i en kø må her lære å utøve fire typer læring, på en gang.
1. Læring av agentens retning mot målet
2. Sosial læring av Cohesion, Separation, Alignment,

Metodene for læring følger gjerne klassisk kontrollteori.

Proporsjonal regulator

Metoden er her basert på kontrolloven

$$Pådrag = (Mål-Posisjon)K$$

Der Pådrag representere et motorpådrag, Mål er et et posisjonsmål, Posisjon er agentens erpoisjon og K er en valgt konstant.

Adaptiv regulator

Den valgte konstanten K kan erstattes an en adaptiv regulator. Dersom en velger å benytte et Neuralt nettverk til å implementere en adaptiv regulator, kan en sette:

Pådrag = NN(Mål-Posisjon)

Der NN() representerer et Neuralt nettverk, som i enkleste tilfelle kan være ett neuron. Dersom en velger å benytte en Genetisk algoritme til å implementere en adaptiv regulator, kan en sette:

Pådrag = (Mål-Posisjon) K(t)

Der K(t) representerer et gen. Agenten kan da opprette en populasjon med K(t) gener. Den velger så i sann tid det genet som gir minst kvadratisk avvik.

FRAMGANGSMÅTE

Framgangsmåte for metoden:

1. Formuler først det eksperimentet agentene skal utføre
2. Formuler så hvordan du vil teste resultatet av forsøket
3. Formuler så metodene agenten skal benytte for å nå mål.
4. Formuler vektleggingen av de sosiale egenskapene med Cohesion, Separation og Alignment

Framgangsmåte med programmering

Her kombinerer du framgangsmåten du benyttet med Bevegelige agenter og framgangsmåten med Sosiale agenter.

OPPGAVE

Kontrollspørsmål om sosiale agenter i kø

1. Gjøre rede for metoder for modellering av køer med sosiale agenter
2. Gjøre rede for hvordan sosiale agenter som køer kan løse oppgaver

Forsøk med sosiale agenter i kø

Det skal gjøres et eksperiment med sosiale agenter i en kø.

1. Lag et landskap der agentene skal passere gjennom et hull i en vegg
2. La målagenten bevege seg gjennom hullet i veggen, slik at den drar med seg de andre
4. Studer hvordan dette eksperimentet påvirker en god balanse for parametrene for kontroll av Cohesion, Separation og Alignment.
3. Kommenter resultatet
4. Hvordan kan en her trene agentens parametre med en Genetisk algoritme?
5. Hvordan kan en her trene agentens parametre med et Neuralt Nettverk?

5.3 Nyetablering av svermer

Spørsmål: Hva menes med nyetablering av svermer?
Svar: Nyetablering er når en sverm agenter beiter på ressurser i ett landskap og flytter seg over til nye ressurser et annet landskap.

Spørsmål: Hva er eksempler på nyetablering av svermer?
Svar: Noen typiske er svermer som representerer markedsmodeller, framvokst av byer, spredning av befolkningsgrupper og endringer i økosystemer.

Spørsmål: Hva kan nyetablering svermer gjøre for deg?
Svar: De kan benyttes til å simulere spredningsforløp under ulike forutsetninger

Spørsmål: Hvordan kan en simulere nyetablering med svermer?
Svar: Et typisk eksempel er mauralgoritmen. Noen agenter tar en tilfeldig tur etter nye ressurser. Den mest heldige etterlater seg mest spor. Deretter følger de andre etter.

LÆRINGSMÅL

Kunnskapsmål

En skal ved avslutningen av denne leksjonen kunne:

1. Gjøre rede for hvordan en kan modellere partikkel svermer
2. Gjøre rede for hva partikkel svermen kan benyttes til
3. Gjøre rede for prinsippene for Partikkel Sverm Optimalisering algoritmen

SYSTEMTEORI

Metasystemet S(t)

Vi har vi et metasystem

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett med agenter, L(t) landskapet og N(t) er relasjoner mellom agenter i landskap. I dette tilfellet er det slik at et sett agenter A(t) er beiter på et sett resurser i landskapet L(t) via et nettverk N(t). Nyetablering svermer er basert på at svermen A(t) søker et nytt sett med ressurser i i landskapet L(t).

Partikkel agenters etikk

Sosiale agenter A(t) som foretar nyetablering har flere sosiale regler. Typiske regler er

$$A(eti,t)=\{\text{Sosiale regler, Søkeregler, Følgeregler}\}$$

Der Sosiale regler er regler som knytter gruppen sammen. Typiske regler er Cohesion og Separasjon avhengig av anvendelsen.

Simulering av passive strømningsmodeller:
Ved simulering av strømningsmodeller har en gjerne:

- Sosiale regler orientert som separasjon
- Følgeregler knyttet til krefter fra eksternt landskap

Simulering av aktive partikler i strømningsmodeller:
Aktive partikler i strømningsmodeller kan f.eks representere objekter i vann. Typiske modeller her er da:

- Sosiale regler orientert om separasjon
- Målorienterte regler for tilpassing av omgivelse
- Følgeregler knyttet til krefter fra eksternt landskap

Simulering av optimaliseringsproblem:
Ved simulering av optimaliseringsproblem har en gjerne:

- Sosiale regler orientert som kohesjon mot optimale områder
- Sosiale regler orientert som separasjon mot lokale agenter

Sosial agent læring

Sosial agent læring er metoder agentene benytter å realisere svermens etikk. Et viktig prinsipp her er at der er ingen sentral kontroll. All læring og kontroll er knyttet til

agentenes individmodeller.

Læring av sosiale regler
Læring av Sosiale regler er basert på estimerting av parametre og kontroll av kraftpådrag mellom bevegelige agenter.

5.2 Partikkel svermer

Spørsmål: Hva menes med partikkel svermer?
Svar: En partikkel agent er en agent som representerer et passivt objekt. En gruppe partikkel objekter opptrer som en partikkel sverm når de forflyttes i et landskap.

Spørsmål: Hva er eksempler på partikkel svermer?
Svar: Partikkel svermer representerer gjerne fysiske objekter. Noen eksempler er flytende objekter i vann, olje, vanndråper og liknende. I agentbasert modellering kan partikler også representere varer, datamaskiner i nettverk, biler, skip og liknende.

Spørsmål: Hva kan partikkel svermer gjøre for deg?
Svar: En typisk oppgave for partikkel svermer er å identifisere strømningsmønster. En annen oppgave er å løse optimaliseringsproblem.

Spørsmål: Hvordan kan en simulere partikkel svermer?
Svar: Et typisk eksempel er mauralgoritmen. Noen agenter tar en tilfeldig tur etter nye ressurser. Den mest heldige etterlater seg mest spor. Deretter følger de andre etter.

LÆRINGSMÅL
Kunnskapsmål
En skal ved avslutningen av denne leksjonen kunne:
4. Gjøre rede for ”Mauralgoritmen” som metode
5. Gjøre rede for hva ”Mauralgoritmen” kan benyttes til

SYSTEMTEORI

Metasystemet S(t)
Vi har vi et metasystem

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett med agenter, L(t) landskapet og N(t) er relasjoner mellom agenter i landskap. I dette tilfellet er det slik at et sett agenter A(t) er beiter på et sett resurser i landskapet L(t) via et nettverk N(t). Nyetablering svermer er basert på at svermen A(t) søker et nytt sett med ressurser i i landskapet L(t).

Sosiale agenters etikk
Sosiale agenter A(t) som foretar nyetablering har flere sosiale regler. Typiske regler er

$$A(eti,t)=\{\text{Sosiale regler, Søkeregler, Følgeregler}\}$$

Der Sosiale regler er regler som knytter gruppen sammen. Typiske regler er

Cohesion og Separasjon. Søkeregler er regler som enkelt agenter benytter for å søke etter nye ressurser. Følgeregler er regler agentene benytter for å følge spor mellom ulike ressurser.

Sosial agent læring
Sosial agent læring er metoder agentene benytter å realisere svermens etikk. Et viktig prinsipp her er at der er ingen sentral kontroll. All læring og kontroll er knyttet til agentenes individmodeller.

5.3 Partikkel sverm optimalisering

Spørsmål: Hva menes med partikkel sverm optimalisering?
Svar: Det er en sverm av partikler som til sammen løser et optimaliseringsproblem. Partikkel sverm optimalisering, er også forbundet med algoritmen Particel Swarm Optimization (PSO), en algoritme som er mye benyttet til å løse optimaliseringsproblem.

Spørsmål: Hva er eksempler på PSO?
Svar: PSO benyttes til optimalisering av kontrollsystem, kommunikasjonssystem, logistikk, kontroll av sanntidssystemer, energiforvaltning osv. Den har altså vis seg å være en smart metode til å løse en lang rekke typer med oppgaver.

Spørsmål: Hva kan partikkel svermer optimalisere for deg?
Svar: En typisk oppgave for partikkel svermer er å identifisere strømningsmønster som er begrenset av lokale forhold. I denne leksjonen skal vi se på hvordan en PSO kan benyttes til å simulere biltrafikk gjennom et komplisert vegsystem.

Spørsmål: Hvordan kan en foreta en partikkel sverm optimalisering?
Svar: Det er egentlig veldig enkelt. PSO er egentlig en variant av algoritmen for sosiale agenter. Kohesjon er byttet ut med et globalt mål, alignment er tatt bort, og separasjon er knyttet til en lokal kostnad.

LÆRINGSMÅL
Kunnskapsmål
En skal ved avslutningen av denne leksjonen kunne:
1. Gjøre rede for ”Partikkel Sverm Optimering (PSO)” som metode
2. Gjøre rede for hva ”PSO” kan benyttes til

Ferdighetsmål
En skal ved avslutningen av denne leksjonen kunne:
1. Gjøre et eksperiment med en PSO algoritme
2. Vurdere egne resultater

SYSTEMTEORI

Metasystemet S(t)
Vi har vi et metasystem

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett med agenter, L(t) landskapet og N(t) er relasjoner mellom agenter i landskap. I dette tilfellet er det slik at et sett agenter A(t) er beiter på

et sett resurser i landskapet L(t) via et nettverk N(t). Nyetablering svermer er basert på at svermen A(t) søker et nytt sett med ressurser i i landskapet L(t).

Agenters etikk
Sosiale agenter A(t) som foretar nyetablering har flere sosiale regler. Typiske regler er

A(eti,t)={Søkeregler}

En partikkelagent har ingen sosiale regler. Det vil si at agenten har ingen felles regler med andre agenter i svermen. PSO svermer har i prinsippet bare to søkeregler. Den ene regelen søker globale mål, den andre regelen søker lokale mål. Det er så en tilfeldig vekting av reglene som bestemmer hva som har størst betydning.

Agent læring
PSO normalt ingen læring i sann tid. Læringen er normalt basert på at en benytter er populasjon med tilfeldige parameter. Deretter ser en hvilke parameter som viser seg å være gode.

Læring med Partikkel Swarm Optimalization (PSO)

PSO algoritmen er basert på en forenkling av Craig Reinholds sosiale regler for Cohesion, Separation og Alignment. Algoritmen er basert på regelen:

$$Vij(t+1) = C1\ Vij(t) + C2\ R1\ [L(best\ local, t)- Lij(t)] + C3\ R2\ [L(best\ global, t)- Lij(t)]$$
$$Xij(t+1) = Xij(t) + Vij(t+1)$$

Der Xij(t) er agent (I,j) posisjonen ved tiden t, Lij(t) en landskap posisjonen, Vij(t) er agentens hastighet i landskapet L(t), [C1, C2, C3] er valgte parametre og [R1, R2] er tilfeldige parametrer mellom [0. 1]. L(best local, t) er beste lokale posisjon og P(best global, t) representerer den beste globale posisjon.

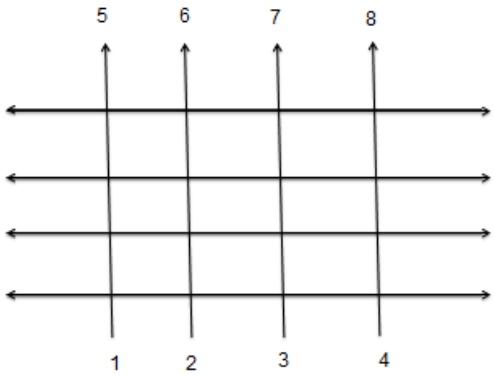
OBLIGATORISK OPPGAVE

Kunnskapsspørsmål

1. Gjør rede for hvordan en kan modellere partikkel svermer
2. Gjør rede for hva partikkel svermen kan benyttes til
3. Gjør rede for prinsippene for Partikkel Sverm Optimalisering algoritmen

Ferdighetsoppgave

Vegsystem med gatekryss



I denne oppgaven skal en benytte PSO algoritmen til å simulere et trafikk mønster i et landskap. Vi tenker oss at vi har et landskap med et vegsystem som vist på figuren, med inngang 1-4, og utgang 5-8. I tilknytting til dette landskapet, ønsker vi å studere hva som vil skje nå partikkel agenter har en tilfeldig inngang 1-4, og en tilfeldig utgang 5-8. I denne simuleringen har altså partikkel agentene sine globale mål, og lokale kostnader som er forbundet med posisjonen i landskapet.

1. Formuler eksperimentet som et lab forsøk
2. Formuler en godhet kriterium for eksperimentet
3. Programmer eksperimentet i Unity, eller et annet språk
4. Gjør rede for resultatet

Litteratur

http://en.wikipedia.org/wiki/Particle_swarm_optimization
http://cswww.essex.ac.uk/technical-reports/2007/tr-csm469.pdf

6 Stasjonære sosiale Agenter

Spørsmål: Hva er stasjonære agenter?
Svar: Stasjonære agenter er agenter med faste posisjoner i et landskap.

Spørsmål: Hva er eksemplene på stasjonære agenter?
Svar: Noen eksempler på stasjonære agenter er agenter som representerer planter, trær, bygninger, industrielle produksjonssystemer osv.

Spørsmål: Hva kan stasjonære agenter gjøre for deg?
Svar: Stasjonære agenter kan løse optimaliseringsproblem. Typiske eksempler er å finne ut optimal ressursutnyttelse, produksjonskapasitet, tidspunkt for optimal vekst, optimal markedspris osv.

Spørsmål: Hvordan kan en modellere stasjonære agenter?
Svar: Stasjonære agenter er normalt konsumenter og produsenter av en ressurs i et landskap. Agentens optimaliseringsproblem er da å tilpasse sine kostnad og produksjonskapasitet i forhold til lokal ressurstilgang.

Læringsmål
Studenten skal ved avslutningen av denne leksjonen kunne:
1. Modellere stasjonære agenter som beiter i et landskap
2. Modellere agenter som utøver en serieproduksjon
3. Modellere agenter som konkurrerer om en ressurs

(Oppdateres: IMRAD, System)

Emner
1. Agent som konsument og produsent
2. Kostnadsfunksjoner
3. Verdi transformasjon
4. Kostnadstilpassing

6.1 Agenter mellom svermer

Spørsmål: Hva er stasjonære agenter?
Svar: Et bevegelig må er mål som endrer seg over tid.

Spørsmål: Hva er eksemplene på stasjonære agenter?
Svar: Typisk .

Spørsmål: Hva kan stasjonære agenten gjøre for deg?
Svar: Det.

Spørsmål: Hvordan kan jeg modellere stasjonære agenter?
Svar: Dersom

Bakgrunn:
Når en sverm agenter er i bevegelse, må agentens hastighet ha omtrent samme retning. Hver agent må da overvåke om den har en hastighet og retning som er i samsvar med midlere hastighet og retning i svermen.

Spørsmål: Hvordan kan hver agent bestemme sin egen hastighet og retning?

Svar: Svaret er at hver enkelt agent må justere sin egen hastighet og retning i forhold til den midlere hastighet og retning i svermen.

Systemteori

Metasystemet S(t)
I dette tilfelle har vi et metasystem

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett med agenter, L(t) landskapet og N(t) er relasjoner mellom agenter i landskap. I dette tilfellet er relasjonene N(t) bestemt av agenten hastighet og retning i forhold til de andre agentene. Agenters etikk A(eti, t) er i dette tilfelle at den tilpasser seg den midlere hastighet og retning i svermen.

Alignment
Læringsmetoden er her basert på at agenten først beregner midlere hastighetsvektor i svermen. Dette kan formuleres som

V(sverm) = Sum(j)(A(j,hast)-A(i,hast))/(Antall-1)

Der V(sverm) er midlere hastighetsvektor, A(i,hast) er hast på egen Agent nr i, og A(j,hast) er hastigheten til alle de andre agentene. Dette må da altså beregnes for alle agentene i svermen.

Avviket mellom agentens hastighet retning og svermens hastighets retning, må omgjøres til et kraft pådrag. Dette kan formuleres som

F(ali)=Gali*||V(sverm)-(A(i,hast)||

Der Gali representere agentens vekt på avviket, og || || representerer retningen på avviket.

Metode i Unity

Opplegget blir det samme som tidligere. Forskjellen blir bare at en må finne en metode, slik at hver agent følger svermens hastighet og retning.

Oppgave

I denne øvingen skal vi opprette en agent sverm, der hver agent overvåker sin egen hastighet og retning, i forhold til alle de andre agenter.

1. Studer algoritmen alignment. Se beskrivelsene i dokument (1) og (2).
2. Studer vedlagte program eksempel UpdateSwarmState() for implementering i Unity.
3. Alignment representerer en egen kraft påvirkning på agenten. Velg en passende forsterkning for agentens separasjon kraftpåvirkning.
4. Opprett en sverm med agenter, med cohesion, separasjon og alignment og studer svermens bevegelse.
5. Sett en felles start og en felles target posisjon, og studer hvordan svermen beveger seg.
6. Juster kraft forsterkning slik at separasjonen setter til et passende nivå.

6.2 Stasjonære Agenter i verdikjede

Spørsmål: Hva er stasjonære agenter i verdikjede?
Svar: Stasjonære agenter i en verdikjede er agenter som videreforedler ressurser fra andre agenter.

Spørsmål: Hva er eksemplene på agenter i verdikjeder?
Svar: Eksempler på agenter i verdikjeder er industrielle næringskjeder, økonomiske systemer og økologiske systemer. I industriell produksjon er det slik at alle som produserer noe, er knyttet til en næringskjede. Det vil si at alle har en eller annen underleverandør av varer, kapital eller tjenester, og alle har en eller annen kunde som mottar den vare eller tjeneste som produseres. Denne serie

av tjenester kan betraktes som en serieproduksjon av verdiøkende tjenester.

Spørsmål: Hva kan agenter i verdikjeder gjøre for deg?
Svar: Agenter i verdikjeder kan løse optimaliseringsproblem. Typiske eksempler er optimalisering av produksjonskapasitet, optimalt tidspunkt for produksjon og optimal prissetting.

Spørsmål: Hvordan kan en modellere en verdikjede av agenter?
Svar: I en verdikjede er alle agenter både konsumenter og produsenter. Samtidig er produksjonen forbundet med en kostnad. Denne kostnaden må legges til som en verdi i produksjonskjeden. Det betyr at alle agentene må optimalisere sin kostnad i forhold til produksjonens verdi på konsumentsiden og på produksjonssiden.

Læringsmål
Studenten skal ved avslutningen av denne leksjonen kunne:
1. Modellere stasjonære agenter som beiter i et landskap
2. Modellere agenter som utøver en serieproduksjon
3. Modellere agenter som konkurrerer om en ressurs

(Oppdateres: IMRAD, System)

Systemteori

Metasystemet i dette tilfellet kan formuleres som:

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett med agenter, L(t) landskapet og N(t) er relasjoner mellom agenter og landskap.

Nettverk N(t)
Nettverket har relasjonen N(t)={A(t),L(t)}. I en serieproduksjon er A(t) A(t)={A(1,t), A(2,t),,,, A(N,t)} et sett agenter der hver agent har en stasjonær tilknytting mellom agent og landskap. Nettverket for kommunikasjon mellom om agenter kan knyttes direkte mellom agenter og via landskap. Dette kan gjøres enklest ved at Agenten A(1,t) konsumerer fra landskap cellen L(1,x1,y1) og produserer til landskap cellen L(2,x1,y1). Agent A(2,t) konsumerer fra cellen L(2,x1,y1) og produserer til cellen L(2,x2,y2) osv.

Landskap L(t)
Vi tenker oss her at landskapet L(t) er representert med en 3D matrise med et sett av landskap L(N,x,y) der L(1,x,y) har en vekstmodell som beskrevet tidligere.

Agentmodell A(t)
I dette tilfelle representer er A(t) et sett agenter

$$A(t)=\{A(1,t), A(2,t),,,, A(N,t)\}$$

der hver agent har samme arkitektur som tidligere. I tillegg er der et identifikasjon nummer for hver agent. Denne identifikasjonen tilordnes agenten ved instanssering i

AgentServer (script).

Agentens tjenester:
A(Sensor,t)={Avhente ressurser fra landskap. Sette produksjon til landskap}
A(Kropp,t)={En enkel geometri}
A(Produser,t)={En lineær trnsformasjon y=a*x, med lineær kortnad}
A(Kontroller,t)={Kontroll av konsume}

Agent etikk A(eti,t)
Hver agent i populasjonen har den samme etikk:
A(eti,t)={Å optimalisere eget konsum i landskap over tid}

Agent etikk A(lær,t)
Hver agent i populasjonen har her også samme metode for kontroll av agentens konsum.

Spillet

Spillet til i dette metasystemet er at hver agent agenten i produksjonskjede, må tilpasse eget konsum etter konsum og produksjon i den foregående agent i produksjonskjeden. Dette forventes å påvirke totalsystemets dynamikk.

Oppgave

Velg passende verdier av referansen R og kontroll K, og opprett er sett med agenter som utøver en serieproduksjon over et landskap.

Spørsmål:
Hvordan blir stabiliteten for konsum og produksjon i verdikjeden?
Hvordan kan en finne optimale verdier av R og K?

6.3 Stasjonære agenter i et marked

Spørsmål: Hva er stasjonære agenter i et marked?
Svar: Stasjonære agenter i et marked er agenter som konkurrerer om en ressurs.

Spørsmål: Hva er eksemplene på stasjonære agenter i et marked?
Svar: Alle økologiske, industrielle og sosiale aktiviteter er forbundet med en eller annen form for parallell konkurranse mellom aktiviteter. I seriebasert produksjon så vi at agentene kunne tilpasse seg ved å tilpasse produksjonskapasitet etter verdier. I en parallell produksjon er dette ikke mulig. Agenten må da tilpasse dens kostnader ved å tilpasse dens kapasitet eller rammevilkår.

Spørsmål: Hva kan stasjonære agenten i et marked gjøre for deg?
Svar: De kan finne en balanse mellom kostnad, kapasitet, pris i forhold til eksterne konkurrenter osv.

Spørsmål: Hvordan kan jeg modellere stasjonære agenter i et marked?
Svar: En kan la agentene ta beslutninger ut fra spillteori.

Læringsmål
Studenten skal ved avslutningen av denne leksjonen kunne:
1. Modellere stasjonære agenter som beiter i et landskap
2. Modellere agenter som utøver en serieproduksjon
3. Modellere agenter som konkurrerer om en ressurs

(Oppdateres: IMRAD, System)

Systemteori

Metasystemet S(t) kan i dette tilfellet også formuleres som:

$$S(t)=\{N(t), A(t), L(t)\}$$

der A(t) representerer et sett med agenter, L(t) landskapet og N(t) er relasjoner mellom agenter og landskap.

Landskap L(t)
Landskap med prismodell er den samme som tidligere. Videre er agentene plassert i hver sin landskap posisjon L(x,y).

Nettverk N(t)
Agentens nettverk N(t)={A(t),L(t)} er i dette tilfelle slik at alle agentene i populasjonen A(t) er forbundet til samme celle L(x,y) i Landskapet L(t).

Agentmodell
Agenter kan modelleres som tidligere med egenskapene

$$A(t)=\{A(ark,t),A(dyn,t), A(eti,t), L(lær,t)\}$$

Der A(ark,t) representerer agentens arkitektur, A(dyn,t) agenten tilstandsdynamikk, A(eti,t) agentens etikk, og L(lær,t) agentens læringsstrategi.

Agentens tjenester:
A(Sensor,t)={Avhente ressurser fra landskap. Sette produksjon til landskap}
A(Body,t)={En enkel geometri}
A(Producer,t)={En lineær transformasjon y=a*x, med lineær kortnad}
A(Controller,t)={Kontroll av produksjons verdi}

Agent etikk A(eti,t)
Hver agent i populasjonen har den samme etikk.
A(eti,t)={Å produsere når den har en verdiøkning}

Agent læring A(lær,t)
I seriebasert produksjon var oppgaven å tilpasse en produksjon volum til en produksjons verdi i samsvar med kostnadsfunksjonen

$$J(t)=Q*X(t)+P*L(t)$$

der X(t) representerer en tilstandsvektor i agenten A(t), Q representerer vektleggingen eller kostnaden på tilstanden, L(t) agentens nettverk til landskapet L(x,y,t) og P ressurstilgang eller kostnader tilknyttet landskapet.

Når flere agenter har parallell tilgang til samme ressurser og verdier, vil bare de agenter som har tilpasset produksjonskostnader, kunne utnytte sin produksjon. Agenten må da lære hva som er riktig tilpasset kostnadsmatrise Q og P i forhold til verdien som ligger i landskapet.

Læring med Genetisk Algoritme
En metode til å finne riktige kostnader er å lære kostfunksjonen med en genetisk algoritme. En typisk metode er da å:

1. La kostnadselementene q1 og q2 være frie gener
2. Generer en populasjon agenter med tilfeldige verdier på gener
3. Ta ut de agentene med beste akkumulerte objektfunksjon
4. Kryss genene fra de beste agentene
5. Generer en ny agent populasjon med nye gener

Spillet
Spillet til i dette metasystemet er at alle agentene må tilpasse sin produksjon kapasitet og kostnader etter pris og kapasitet i markedet. I dette spillet forventes det at det der blir en mer komplisert dynamikk, påvirket av antall agenter som overlever i markedet.

Oppgave
Opprett et sett agenter i en parallell produksjon der hver agent regulerer sin egen produksjonskapasitet i samsvar med en prismodell og en kostnadsmodell.

Spørsmål:
Hvordan blir stabiliteten til pris og produksjon?
Hvordan blir stabiliteten til agentenes kostnadsfunksjoner?

Biological agents
1. Fish swarm: http://www.youtube.com/watch?v=UM8SzF6_0sM&feature=related
2. Ant agents: http://www.youtube.com/watch?v=C1XYMG5R5cQ&feature=related

6.4 Horisontal evolusjon
Bakgrunn:
Agenter som opptrer i en kø, har mange frihetsgrader. Optimal kontroll av agentens egenskaper er da bestemt av egenskapen ved ressursen, og svermens størrelse. Detter er igjen noe som kan variere over tid. Vi ser da at der er ikke enkelt å identifisere optimal agent masse eller sverm størrelse.

Spørsmål: Hvordan kan individbaserte agenter, bli intelligente, slik at agentene lærer å optimalisere sine egne tjenester?

Svar: Svaret er at de kan forbedre sine egne tjenester ved å optimalisere en kostnadsfunksjon.

Metode

Metasystemet S(t)
I dette tilfelle har vi et metasystem

S(t)={N(t), A(t), L(t)}

der A(t) representerer et sett med agenter, L(t) landskapet og N(t) er relasjoner mellom agenter i landskap. I dette tilfellet er relasjonene N(t) bestemt av agenten posisjon i forhold til de andre agentene. Agenters etikk A(eti, t) er i dette tilfelle at den vil optimalisere sine egenskaper i forhold til en kostnadsfunksjon.

8 Referanser

1. Yndestad H. 2010. Agents and landscapes in Complex Systems. Kompendium. Høgskolen I Ålesund.
2. Kopi av bok om Boids (Ligger I mappen på Fronter)
3. Robin Bye: Unity3D Tutorial. Fireball
4. Craig Rainholds: Steering Behaviors For Autonomous Characters. Artikkel
5. Ira Rudowsky: Intelligent Agents. Artikkel

9 Vedlegg: Program Script Eksempler

Hallo Agent

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class Agent : MonoBehaviour {
// Global data variable
public int myNr;                    // Agent Number
public string myName="Bond";// Agent Name
public float waitTime=1.0f;      // Agent wait time
public AgentServer controller; // Connection to AgentServer

        // Start process
        void Start() {
                // Agent start methods
                StartCoroutine(UpdateState());
        }           // End Start

        // Runtime process
        IEnumerator UpdateState() {
         while(true) {
         // Agent runtime methods
        ReadMe();
        yield return new WaitForSeconds(waitTime);
            } // End while
        } // End Update States

        void ReadMe() {
        Debug.Log("Hello, my name is "+myName+" Nr "+myNr);
        }// End Start
}
```

Hallo Agent Server

Kode for AgentServer(script)

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class AgentServer : MonoBehaviour {
// Global variable
public Agent aPrefab;            // Agent prefab connection
// A list of Agents
public List<Agent> Agents=new List<Agent>();
public int agentNr;                        // Agent number
public int maxAgents=10;                   // Max agents in collection
public float waitTime=2.0f; // Wait time in seconds
        void Start () {
                // Stat methods
                CreateAgents();     // Create a set of agents
                CreateAgents();
                StartCoroutine(UpdateState());
        }

        IEnumerator UpdateState() {
        while(true) {
        // Server methods
        yield return new WaitForSeconds(waitTime);
        } // End while
        }// End Update States

        void CreateAgents () {
                for (int i = 0; i < 10; i++){
                // Creates an agent at current position and rotation
                Agent anAgent = new Agent ();
                anAgent = Instantiate(aPrefab, transform.position, Quaternion.identity) as Agent;
                anAgent.myNr=i;
                // Set agents to a random position inside the swarm sphere
                anAgent.transform.parent = transform;              // Set agent parent position
                anAgent.transform.localPosition = new Vector3(     // Set agent local position
                Random.value * collider.bounds.size.x,             // Set random bound x-siomze
                Random.value * collider.bounds.size.y,             // Set random bound y-size
                Random.value * collider.bounds.size.z) - collider.bounds.extents;
                Agents.Add(anAgent);                               // Set agent to agent list PUT
                } // End for
        } // End Create Agents

        void ReadMe() {
                Debug.Log("Read nr: ");
        }       // End Start
} // End program
```

Update: Swarm State

```
void UpdateSwarmState() {
// --- Reset Variable ----
        cohCen=Vector3.zero;
        Vgro = Vector3.zero;
        sepDis = Vector3.zero;
        //tarDis= Vector3.zero;
        N=0;
        foreach (GeneralAgent anAgent in controller.myAgents) {
                if (anAgent == this) {   // To all agents exept to this agent

                }// End if
                else {
// === COHESION ================================
                // --- Sum of Agent positions
                cohCen = cohCen+anAgent.transform.localPosition;

// === SEPARATION ==============================
                // --- Sum of Agent distence
                // --- Distance to next agent
```

```
                //relDis= myPosition-anAgent.transform.localPosition;
                relDis= X-anAgent.transform.localPosition;

                // --- Distance adjusted
                relDis = relDis/(relDis.sqrMagnitude+0.1f);
                // --- Sum of relative distances
                sepDis = sepDis+relDis;

// === ALIGNMENT =================
                // --- Sum of Agent speed
                Vgro=Vgro+anAgent.rigidbody.velocity; // Speed sum
                N=N+1;  // Agent numbers in swarm
                }       // End Eles
        }       // End Foreach Agent

// === COHESION ======
        cohCen=cohCen/(N-1);     // Mean Cohesion Position
        cohDis=(cohCen-X);       // Distance to Cohesion Centre
        cohDir=cohDis.normalized;// Direction to Cohesion Centre

// === SEPARATION ====
        //sepDis=sepDis/(N);

// === ALIGNMENT ====
        Vgro=Vgro/(N-1);                        // Mean Agent Speed
        aliDir=Vgro.normalized;          // Mean Adent Speed Direction
}// End UpdateMyFlockState
```

Time Server

```
using UnityEngine;
using System.Collections;
public class TimeServer : MonoBehaviour {
        // Time values
    public int i;
    public float simTime, playTime, aTimeScale, curTime, offsetTime;
    public float StartTime;
    public float [] RealTime, SimTime,  EndTime;
    public string [] TimeTxt; // (sec, min, hr, day, yr)
    public float dTime=1.0f;
        private Rect _win1Rect;

        // Window values
        private float maxTimeSpeed=100.0f;
        private float minTimeSpeed=0.0f;
        private float scrollPos=1f;

        void Start() {
                setStartTime();                         // Sett timme init
                setGUI();                               // Set and Start GUI
                StartCoroutine(startClock());                   // Start clock process

        }  // Set all start conditions

        void Update() {
                //countFrames();
                //countTime();

        }  // Computes for each frame


        void setStartTime() {
                TimeTxt = new string [] {"sec","min", "hr", "day", "yr"} ;
                RealTime =new float [] {0,0,0,0,0} ;
                SimTime =new float [] {0,0,0,0,0} ;
                aTimeScale=1f;
                playTime=0f;
                StartTime=Time.time;
        }       // set start time condition

        void setGUI() {
                // Set GUI menues (x1,y1,x2,y2)
                _win1Rect = new Rect(20,20,120,160); // Set GUI window
        }       // set GUI for time control

        IEnumerator startClock() {
            while(true) {
                // Update my Clock state
                playTime=playTime+dTime;        // Uptate playtime
```

```
                countTime();                                    // Update simtime
                yield return new WaitForSeconds(dTime);
            } // End while
        }       // Time control window

        void countTime() {
                curTime=Time.realtimeSinceStartup;
                simTime=aTimeScale*playTime;   // Simtime in seconds
                SimTime[3]=(int)(simTime/(24f*60f*60f) %24f);       // Days
                SimTime[2]=(int)(aTimeScale*playTime/(60f*60f) %60f);                   //
Hours
                SimTime[1]=(int)(aTimeScale*playTime/60f %60f);                         //
Minutes
                SimTime[0]=(int)(aTimeScale*playTime %60f);                             //
Seconds
        }       // End countTime

        void OnGUI() {
                // Draw a window
                string guiHeadLine="Time Scaling";
                _win1Rect=GUILayout.Window(1,_win1Rect,DrawWin1,guiHeadLine);
        }       // End OnGUI

        void DrawWin1(int winId) {

                float oldScrollPos=scrollPos;
                // Get window scroll position

        scrollPos=GUILayout.HorizontalScrollbar(scrollPos,1,minTimeSpeed,maxTimeSpeed);

                aTimeScale=scrollPos;
                // Set information to the window
                GUILayout.Label("Play Time: "+playTime);
                GUILayout.Label("Time Scale: "+aTimeScale);
                GUILayout.Label("Sim Time: ");
                GUILayout.Label("   "+"day, hr, min, sec");
                GUILayout.Label("   "+SimTime[3]+"       "+SimTime[2]+"
"+SimTime[1]+"     "+SimTime[0]);
                GUI.DragWindow();       // Display window information
                if(scrollPos != oldScrollPos) {
                        // Scaling the internal simulation timer
                        Time.timeScale=aTimeScale;
                }  // End Time scaling
        }  // End DrawWintranslation

} // End program
```

Whacher

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
//      === Whacher     ============
//      Processes
//      Input
//              Target
//      Output
//              my ttransform
//  Project: Virtual Mbre
//      Autors:
//      Original authors: Unify Community and
//  Benoit Benoit FOULETIER,
//  Modified by: R. Bye
//      Modified by H.Yndestad
//  Last update 09.10.2009
//      By H. Yndestad
//      ============================
public class Watcher : MonoBehaviour {
        public AgentSystem aTarget;
        //public Target aTarget;
        public Vector3 myTarget,flockCenter;
        void LateUpdate() {
                if (aTarget) {
                        // my posision = flockcentre pos + target pos
                        //myTarget = aTarget.flockCenter+aTarget.transform.position;
                        myTarget=GameObject.Find("Target").transform.position;
                        flockCenter=aTarget.flockCenter;
                        //myTarget = (aTarget.flockCenter+aTarget.transform.position);
                        //myTarget = (aTarget.flockCenter);
                        myTarget = aTarget.cameraPos;
                        transform.LookAt(myTarget); // Update my position
                        //transform.position(myTarget); // Update my position
                        //print("Camera Target = "+ myTarget);
```

```
                }       // End if
        }       // End LateUpdate
}  // End Watcher
```

Move In Circle

```
using UnityEngine;
using System.Collections;

public class MoveInCircle : MonoBehaviour {

        // Move target around circle with tangential speed
        public float tangentialSpeed; // m/s
        public float circumference;    // m
        public float targetRadius;              // m
        public float period;                    // s
        public float angularSpeed;              // rad/s
        public float currentAngle;              // rad/s
                Vector3 Centre=new Vector3();

        // Use this for initialization
        void Start () {
                tangentialSpeed = 6f;
                circumference = 600f;
                targetRadius = circumference/(2*Mathf.PI);
                period = circumference/tangentialSpeed;
                angularSpeed = 2*Mathf.PI/period;
                currentAngle = 0f;
                Centre=new Vector3(1000,280,1000);
        }


        // Update is called once per frame
        void Update () {

                transformlocalPosition = Centre+ targetRadius*(
                        new Vector3(Mathf.Sin(currentAngle), 0,
Mathf.Cos(currentAngle)));

//              rigidbody.velocity = new Vector3(Mathf.Cos(currentDeg), 0, -
Mathf.Sin(currentDeg));

                currentAngle += angularSpeed*Time.deltaTime;

                if (currentAngle > 2*Mathf.PI) {
                        currentAngle = currentAngle-2*Mathf.PI;
                }
        }
}
```

Land Server

```
using UnityEngine;
using System.Collections;
public class LandSystem : MonoBehaviour {
// --- External Network ---
public TimeServer aTimeServer;
public float dTime;
//public ClimateServer aClimateServer;
//public SunServer aSunServer;

//public SeaServer aSeaServer;

// --- Abstract Landscapes ---
// --- Physics data ----------------
public float [] SunRad;                 // Sun Radiation
public float [] AirTemp;        // Air temperature
public float [] AirRain;
public float [] AirWind;        //
public float [] AirO2;                  //
public float [] AirCO2;         //
public float [] SeaRad;         //
public float [] SeaTemp;        //
public float [] SeaO2;                  //
public float [] SeaCO2;         //
public float [] SeaRain;        //
public float [] Risc;           //
public float [] Cost;           //

// ---- Resources ----------
public float [,,] Lres;                 // Landscape resource
public float aRes,bRes,maxRes;          // Resources parametre
```

```
public float [,,] L;                    // (lon,lat,hight)

        // Use this for initialization
        void Start () {
                Initiate();
                StartCoroutine(UpdateState());
        }  // End Start

        void Initiate() {
                AirTemp =new float [] {0f,0f,0f,0f,0f} ;
                AirRain =new float [] {0f,0f,0f,0f,0f} ;
                AirWind =new float [] {0f,0f,0f,0f,0f} ;
                AirCO2 = new float [] {0f,0f,0f,0f,0f} ;
                SeaTemp =new float [] {0f,0f,0f,0f,0f} ;
                SeaCO2 = new float [] {0f,0f,0f,0f,0f} ;

                // Create a landscape area
                L = new float [100,100,100];
                Lres = new float [100,100,100];
                Lres[10,10,10]=10.0f;

                dTime=1.0f;                             // Start time
                // ---- Set growth resource parametre
                aRes=0.01f; maxRes=1000.0f;
                bRes=aRes/maxRes;
        }       // End initiate

        void printMe() {
        // --- Agent State=X[pos,vel,accel,force,mass,friction,vol,Kost]
                Debug.Log("AbstrLandscape resource: "+Lres[10,10,10]);
        }       // End


        IEnumerator UpdateState() {
                // --- Curret Agent State ---
                while (true) {
                        UpdateLandState();
                        printMe();
                        dTime = Random.Range(1f, 2f);
                        yield return new WaitForSeconds(dTime);
                }  // End while
        }       // End dynamic state

        void UpdateResourceState() {
                // --- AIR LANDSCAPE ---
            Lres[10,10,10]=(1.0f+aRes*dTime)*Lres[10,10,10]-
bRes*dTime*Lres[10,10,10]*Lres[10,10,10];
                // Cost;        //

    } // End Update Air Temp state

        void UpdateLandState() {
                // --- AIR LANDSCAPE ---
                UpdateResourceState();                                  // Update resource
state
                //SunRad;       // Sun Radiation
                //AirTemp[0]  =aClimateServer.Xatmp[0];         // Air temperatur
                //AirRain[0]  =aClimateServer.Xrai[0]; // Air raib
                //AirWind[0]  =aClimateServer.Xwin[0]; // Air wind
                //AirCO2[0]   =aClimateServer.Xco2[0]; // Air c02
                // --- Sea Landscape ---
                //SeaRad;       //
                //SeaTemp[0] = aClimateServer.Xstmp[0];         // Sea temp
                //SeaCO2[0] = aClimateServer.Xco2[0]; // Sea CO2
                //SeaO2;
                // Risc;        //
                // Cost;        //

    } // End Update Air Temp state

}
```

Agent Server

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class AgentSystem : MonoBehaviour {
//   === FlockingAgent  ===
//   Purpose: Manage an agent flock
//              Create a set of agen populations
//      Compute flocking centre anv velocity
```

```
//      Manage a flocking path way
//              Update landscape to agents
//      Processes
//              geneUpdate
//              stateUpdate
//              landUpdate
//      Input
//              Landscape
//              FlockingTarget
//      Output
//              This to Agents
//      Autors
//  Last update 09.10.2009
//      By H. Yndestad
//      ============================

//  === The Marine Game ===
//      --- Create landscapes
//      Create sun radiation landscape
//      Create temperature landscape
//  Create salinity landscape
//      --- Create genes
//      Plankton genes
//      Zooplankton genes
//      --- Create populations
//      --- Updates
//      Update population state
//      Update landscape
//      Update genes
//      === END GAME ==============

        public TimeServer aTimeServer;

//      === LANDSCAPE ===
//      --- Sea physics ---
        internal float [] myTerrain; // (x,y,z)
        internal float [] seaTemp; // (x,y,z)

// --- Air Physics ---
        internal Vector3 sunRad, sunPos; // (x,y,z)
        internal float [] airTemp; // (x,y,z)

//      === AGENT ===
        public Agent prefab;
        public List<Agent> agents = new List<Agent>();
        public int AgentId;             // (Type,,,,,,)
        public int AgentType;  // (Type,,,,,,)

// --- Flocking references ---

//      === Flocking Genes ===
//      --- Flocking Id Genes ---
        internal int myId;              // (nr, sex)
        internal int myType;   // (marin, )

// --- Flock Ethics Genes ---
        public Transform target;
        internal float AgentTypeTarget; // (Type,,,,,,)
        internal float AgentPosTarget; // (Type,,,,,,)
        internal float FlockPotentials; // (Speed,Place,,,,,)

// --- Dynamics Genes ---
        internal float dTime,simTime;
        internal Vector3 myPosition, mySpeed, myScale, aPosition, aSpeed;
        public float [] RealTime, SimTime,  EndTime;
        public float waitTime;

//      --- Flocking Genes Dynamics ---
        internal float [] FlockGeneDynamics;
        public float minVelocity = 1;
        public float maxVelocity = 8;
        public int   flockSize=1000;
        public float centerWeight = 1;
        public float velocityWeight = 1;
        public float separationWeight = 1;
        public float followWeight = 1;
        public float randomizeWeight = 1;
        public Vector3 flockCenter;
        public Vector3 flockVelocity;
        public Vector3 center;
        public Vector3 velocity;

        public Vector3 cameraPos;

// LandscapeSensor(rad, water, salin,soil,,,)
// LandscapeCost(c1,,,,,,,,,cn)
```

```
        internal float [] Landscapes;          // (Landscape matrix)
        internal float [] sensors;             // (Landscape identification vector)
        internal float [] LandscapeCost;// (Landscape priority)

// ============================================

//GameObject AgentSystem1;

void Start() {
                SimTime =new float [] {0,0,0,0,0} ;            // Start the game
                createGenes();          // Create a set of genes
                createLandscape();      // Update landscape
//              AgentSystem1.transform.position=Vector3(0.0f,0.0f,0.0f);
                GameObject.Find("AgentSystem").transform.rotation =
Quaternion.identity;
                GameObject.Find("AgentSystem").transform.position =  Vector3.zero;
                GameObject.Find("AgentSystem").transform.localScale = Vector3.one;
                StartCoroutine(landUpdate());
                createFlock();          // Create an Angent Flock population
        }       // End start

        void createFlock() {
                // Create an Angent Flock population
                // (By a gene parameter)
                for (int i = 0; i < flockSize; i++){
                // Creates an agent at current position and rotation
                        Agent agent = Instantiate(prefab, transform.position,
transform.rotation) as Agent;
                        agent.transform.parent = transform;                    // Set
agent parent position
                        agent.transform.localPosition = new Vector3(// Set agent local
position
                        Random.value * collider.bounds.size.x,                 // Set
random bound x-siomze
                        Random.value * collider.bounds.size.y,                 // Set
random bound y-size
                // Set random z-size, relative to collider z-size
                        Random.value * collider.bounds.size.z) -
collider.bounds.extents;
                // --- Set Agent parametre ------------
                        //agent.Aid[1]=(int)(10*Random.Range(0.0f,0.3f));
                        //agent.Aid[2]=(int)(10*Random.Range(0.0f,0.2f));
                        agent.controller = this;                                        //
Set this data to agent controller
                        createGenes();
                        agents.Add(agent);
        // Set agent to agent list
                }       // End For
        }       // End create Agent flock


        void createGenes() {
                // Create a sett of agent genes
                //int AgentId=1;
                //float flockSize = Random.Range(10f, 90f);
                // Set genes to the agent list
        }       // End start

        void createLandscape() {
                // Create a sett of agent genes
                // Set genes to the agent list
        }       // End start

        void UpdateTime(){
                //simTime=aTimeServer.SimTime[1];
                simTime=Time.realtimeSinceStartup;
                //float simTime=aTimeServer.SimTime[0];
                //float waitTime = Random.Range(0.2f, 1.5f);
                waitTime = 5f;
                //print("simTime = " +simTime);
        }  // End Update


        IEnumerator landUpdate() {
                // Read new landscape state
                // Set new landscape to agents
                while (true) {
                        UpdateFlock();                                                  //
Update flock state
                        UpdateSunState();                                               //
Update Suns States
                        UpdateTime();
                        yield return new WaitForSeconds(waitTime);
                }       // End while true
        }       // End landUpdate
```

```
        void UpdateCameraPos(){
                cameraPos=sunPos;
                //print("Sun Rad = " +sunRad);
        }  // End Update

        void UpdateSunState(){
                Vector3 sunRad = new Vector3(20, 20, 20);
                sunPos=new Vector3(500, 50, 200);
                sunRad.y= 20+20*Mathf.Sin(2*3.14f*simTime/(60));
                //print("Sun Rad = " +sunRad);
        }  // End Update

        void UpdateFlock(){
                // Update Flock center and velocity
                foreach (Agent agent in agents){
                        center += agent.transform.localPosition;
                        velocity += agent.rigidbody.velocity;
                        }  // Updates center and velocity
                flockCenter = center / flockSize;
                flockVelocity = velocity / flockSize;
        }  // Update Flock

}  // End Program
```

Agent

```
using UnityEngine;
using System.Collections;
//using System.Collections.Generic;

public class Agent : MonoBehaviour {
//      === Agent Game ============
//      --- Agent 1 Etichs ---
//              1. Landscape (Terranin, Sun Radiation, Flock)
//              2. Cost (+/-Terrain, - Sun Radiation, +/-Flock, spawning)
//              2. Growth(+/- Cost, growthrate)
//              3. Recruitmen (agent sex, Cost, spawning)
//      Agent 1 Learning
//              1. Position (Recruitment) (Random by landscape)
//              2. Growth (Position, cost)(Genes)
//              3. LifeCycle(spawning, mortality)(Genes)

// --- Agent 2 Ethics ---
//              1. Landscape (Sea volume, Agent 1)
//              2. Cost (+Movement, -Target +spawning)
//      Agent Learning
//              1. Position(Target tracing)
//              2. Growth(cost)(genes)
//              3. LofeCycle(spawning, morttality)(gene)

//      Input
//              AgentSystem(Landscape, agentState, flockingState)
//              TimeServer (TimeState)

//      Output
//              LandScape (agentState)

//  Project: Virtual More
//      Autor: H. Yndestad
//  Last update 17.11.2009
//      ============================

// --- Array dimension -------------
    internal int dim=9;                                             // Array dimensions
// --- Time data ---
    internal float dT;                                          // Delta time

// ==== Agent Format ===========
// --- Aid = agent[nr, type, sex, spawn, value]
// --- Agent State=X[pos,vel,accel,force,mass,friction,vol,colour]
// --- Agent Force=F[sum gravity,friction]
        public int [] Aid = new int[9];                 // Agent identity vector
    internal Vector3 [] X=new Vector3[9];     // Agent state vector
    internal Vector3 [] Xp=new Vector3[9];    // Past state vector
    internal Vector3 [] Xe=new Vector3[9];    // Error state vector
        internal Vector3 [] F=new Vector3[9];         // Error state vector

// ====== Landscapes Format ===========
// Lph = physics[terrain, sunrad, temp]
// Lfl = flock[mean, aligment, sparation]
// Lab = abstract[type, pos, kost]
// Lag = agent[type, pos]

// --- Landscape=[ter,sunPos,sunRad]
```

```
    internal Vector3 [] L=new Vector3[9];     // Landscape vectors
    internal Vector3 [] Lp=new Vector3[9];    // Past Landscape vector
        internal Vector3 [] Lph=new Vector3[9]; // Ag Landscape
        internal Vector3 [] Lfl=new Vector3[9]; // Flocking Landscape
    internal Vector3 Vel=new Vector3();
// --- Targets ---
// Tph = physics[terrain, sunrad, temp]
// Tab = abstract[type, pos, kost]
// Tag = agent[type, pos]

// Target=[position]
    internal Vector3 [] T=new Vector3[9];     // Target Vector
        internal Vector3 [] Tp=new Vector3[9];        // Paste Target Agent data
        internal Vector3 [] Tab=new Vector3[9]; // Abstract Landscape Target
        internal Vector3 [] Ta=new Vector3[9];        // Agent Target
        internal Vector3 [] Tph=new Vector3[9]; // Physics Tareget

// --- Id ---
// Apo = position[pos, velos, accel, force]
// Abo = body[energy, mass, vol, friction, colour]

// === Genes ======================================
// Gtr = tracing[Leanrate]
// Gre = recruit[maturity, recruit, mortality]
// Genes=[posRate, growRate, flockAlignRate, flockSepRate]
        internal Vector3 [] G=new Vector3[9];         // Genetic Agent vectors
        internal Vector3 [] Gp=new Vector3[9];        // Past Gene vector state
        internal float []   Gfl=new float[9];  // Flocing Gene vectors
        internal float []   Gtr=new float[9];         // Tracking Gene vectors
        internal float []   Gre=new float[9];         // Recruit Gene vectors

// === FLOCKING FORMAT ================================
// Flocking = [flockTargetPos, flockMeanPos,flockVeloc, relativPos, separation]
    internal Vector3 [] Fl=new Vector3[9];    // Flocking vector state
        internal Vector3 [] Fp=new Vector3[9];        // Flocking vector state
        internal Vector3 g=new Vector3();             // Gravity force vector

//  === COST FUNCTIONS ============================
        internal Vector3 [] J=new Vector3[9];         // Agent object function vector
        internal Vector3 [] Q=new Vector3[9];         // Agent cost vector

        internal Vector3 [] K=new Vector3[9];         // Agent force gain kontrol

        //public Transform target;     // Flocking references


    // ================================================
        public TerrainData terr;
        internal AgentSystem controller; // AgentSystem copy

        void Start() {
                SetMyIdentity();
                SetMyParametre();
                StartCoroutine(updateState());
                //setMyGenes();
        }       // End Start

        void SetMyIdentity() {
                // Aid = agent[nr, type, Target]
                //Aid[0]=   ;
                //Aid[1]= Type:  ,  ,  ,  ,  ;
                //Aid[2]= Target:0=Land position, 1=Sun, 2= Moveing target;
                Aid=new int[dim];// Id vector: {Nr, type,sex}
                Aid[1]=(int)(10*Random.Range(0.0f,0.3f));
                Aid[2]=(int)(10*Random.Range(0.0f,0.2f));
        }       // Set my identity

        void SetMyParametre() {
                for (int i=0; i<dim; i++) {
                        X[i] = 10.0f*(new
Vector3(Random.value,Random.value,Random.value));
                        Xp[i] = new Vector3(Random.value,Random.value,Random.value);

                        L[i] = new Vector3(Random.value,Random.value,Random.value);
                        Lp[i] = new Vector3(Random.value,Random.value,Random.value);

                        T[i] = new Vector3(Random.value,Random.value,Random.value);

                        Tp[i] = new Vector3(Random.value,Random.value,Random.value);

                        G[i] = new Vector3(Random.value,Random.value,Random.value);
                        Gp[i] = new Vector3(Random.value,Random.value,Random.value);


                        F[i] = new Vector3(Random.value,Random.value,Random.value);
                        Fp[i] = new Vector3(Random.value,Random.value,Random.value);
```

```
                        // Gtr = tracing[Leanrate, ContrGain]
                        Gtr[i]=Random.value;      // Tracking genew
                        Gfl[i]=Random.value;   // Flocking gene vector
                        // Gre = recruit[maturity, recruit, mortality]
                        Gre[i]=Random.value;   // Recruitment gene
                        // Set control gene vector
                        // Set cost gene vector
                }  // End for
                // Gravity force
                //F[1]=Vector3.zero;
                F[1].y=-9.81f; // F=-g*M
        }       // End target identification


        void updateGenes() {
                // Update genes when mature
                // Crossing genes wwith the best
                // Mutate genes
        }       // End

        void updateLifeCycleState() {
                // Update genes when mature
                // Crossing genes wwith the best
                // Mutate genes
        }       // End

        void printMe() {
                // --- Agent State=X[pos,vel,accel,force,mass,friction,vol,Kost]
                Debug.Log("I= "+Aid[1]+" X= "+ X[0]+" V= "+X[1]+" A= "+X[2]+" F=
"+X[3]+" M= "+X[4]+" Fric="+X[5]+" J= "+X[7]);
        }       // End

        IEnumerator updateState() {
                // --- Curret Agent State ---
                while (true) {
                        dT=Time.deltaTime;                                      // Get time change
                        ReadTarget();                                           // Update landscape
state
                        IdentAgent();                                           // Update agent
state
                        flockingControl();                                      // Update flocking
state
                        trackTarget();                                          // Compute agent
control
                        updateCost();                                           // Update cost
function
                        //printMe();

                        // dT etter samplingsteoremet og avstand
                        float waitTime = Random.Range(0.2f, 1.5f);
                        yield return new WaitForSeconds(waitTime);
                }  // End while
        }       // End dynamic state


        void ReadTarget() {
                // === Landscapes ===
                getLandscape(); // Read terrain position
         // --- Sun Landscape ---
                L[1]=controller.sunPos;                 // Sun Position
                //Lph[1]=new Vector3(500,50,200);
                L[2]=controller.sunRad;                         // Sun Radiation
                // --- Targets ---
                Ta[1]=controller.target.localPosition;          // Position of a moving
target
        }       // End Indent Landscape

        void getLandscape() {
                int ySize=600;
                L[0]=new Vector3(325f,1f,200f); // Set position
                // Read terrain hight
                L[0].y = terr.GetHeights(325,200,1,1)[0,0] * ySize;
         //print("L[0] "+ L[0]);
        }       // End Indent Landscape


        void IdentAgent() {
                // === Agent states ===
                // Apo = position[pos, velos, accel]
                //Apo[0]=transform.localPosition;       // Get agent old position
                //Apo[1]=rigidbody.velocity;                    // Get agent old velocity
                X[0]=transform.localPosition;       // Agent New position
                //X[1]=rigidbody.velocity;              // Agent New velocity
        }       // End Ident Agent
```

```
        void trackTarget() {
                // === Tracing Algorithm ============
                // --- Target selection ---
                float Lr=Gtr[0];                        // Learning rate gene
                Vector3 R= new Vector3();
                if (Aid[1]==0) {
                        R=L[0]; // Ground Target
                } // End Ground target

                if (Aid[1]==1) {
                        R=L[2]; // Sun radiation
                        //Debug.Log("Id= "+Aid[1]+" J= "+ X[8]);
                } // Ende sun position

                if (Aid[1]==2) {
                        R=Ta[1]; // Target
                } // End select target

        // === Target Tracking Method ===
                // --- Agent State=X[pos,vel,accel,force,mass,friction,vol,Kost]
                // Tracking algorithm
                // Disanxe Error:       E=(T-X)
                // Force Control:       F=K*E-KvV
                // Acceleration:                A=F/M + g
                // Speed:                       V=V+A = V + F/M
                // Distance Target Error
                Xe[0]=Lr*Xe[0]+(1-Lr)*(R+Lfl[0]-X[0]); // Estimated force
                float K=1.0f;               // Control gain
                F[0]=Xe[0]*K;               // Force gain
                float Kv=X[5].y*0.1f;               // Resist coef
                F[0]=F[0]-X[1]*Kv;          // Resistance force
                // Gravity
                Vector3 Fg=new Vector3(0,-9.81f,0);
                float M=X[4].y;
                F[0]=F[0]+0.1f*Fg*M;                // Add gravity
                // Force=Ferror+Fgravity-Fresist
                //Vel=Lr*Vel+(1-Lr)*F[0]/M;
                //rigidbody.velocity=Vel;
                rigidbody.velocity=F[0];
        } // End agent


        void updateCost() {
        // --- Agent State=X[pos,vel,accel,force,mass,friction,vol,Kost]
        // === Estimate Movement Force ===
                // V(n)=(X(n)-X(n-1))/dT
                // A(n)=(V(n)-X(n-1))/dT
                // F(n)=M*A(n)
                  X[1]=(X[0]-Xp[0])/dT;             // V(n)=(x(n)-x(n-1))/dT
                  X[2]=(X[1]-Xp[1])/dT;             // a(n)=(v(n)-v(n-1))/dT
                  X[3]=X[2]*X[4].y;                 // F(n)=a(n)*m
                  // Friction forvce=
                  // Gravity force =

        // --- Estimate Cost function ---
                // Object fubction
                // J = XQX          ｵX+ LPLｵX

                  float le=0.95f;
                  X[7]=le*X[7]+(1-le)*X[3]*dT;          // J(n)=J(n)+F(n)*dT
                // J = J + Q*Fm*dT + Q*L
                // Update energy input from target
                // E = R*T
                Xp=X;   // Update past state
        }       // End


        void flockingControl() {
                // === Separation: Agent separation distance ===
                Vector3 separation = Vector3.zero;
                foreach (Agent agent in controller.agents) {
                        if (agent == this) {   // To all agents exept to this agent

                        }       // End if
                        else {
                                // Compute distanse to other agents
                                Vector3 relativePos= transform.local Position-
agent.transform.local Position;
                                // Accumulate separation index separation =
relpos/sqr(relpos)
                                separation += relativePos/(relativePos.sqrMagnitude);

                        } // End else
                }       // End foreach
```

```
                //  === Flock target identifications ===
                //Lf[1]=controller.target.localPosition;       // Position of a moving
target
                Lfl[1]=controller.flockVelocity;                        // Mean flock
velocity
                //Lfl[2]=separation;                                                    //
Separation in speed
                Lfl[2]=separation*dT;                                           //
Separation in position

                //  === Flocking Gene Control  ====
                Lfl[0]=Gfl[1]*Lfl[1]+Gfl[2]*Lfl[2];
                //totalFlockDist=Lf[0];
          //print(Lfl[0]);
                //  ---  Follew flocking target
          }//  End flocupdate

}         //  End Agent
```