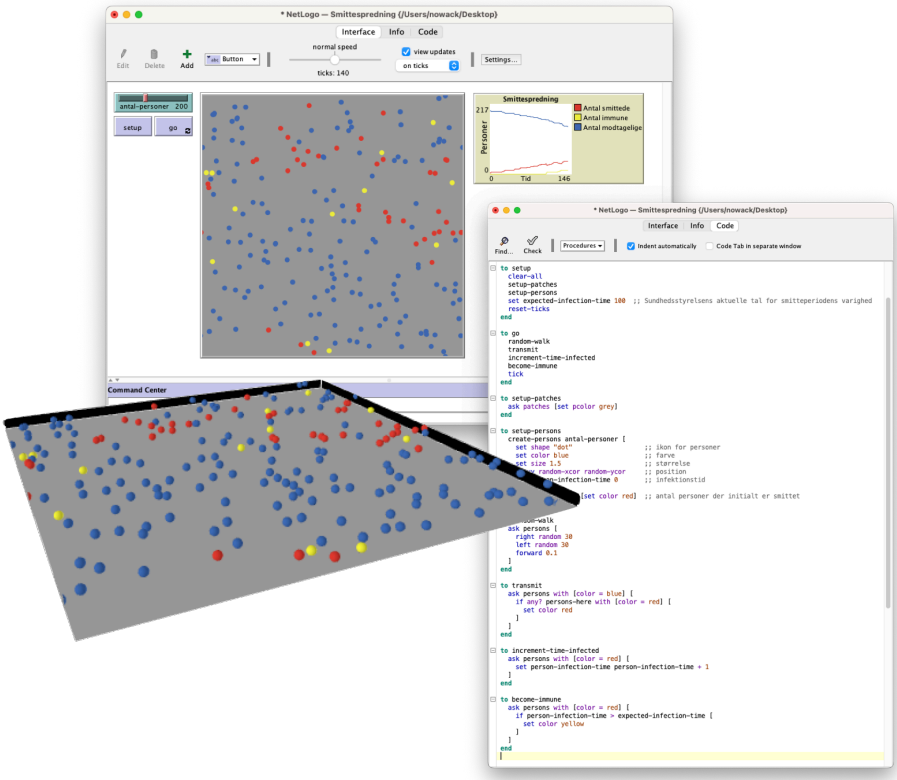
Introduktion til agentbaseret modellering med NetLogo

Version 1.1



"Without abstraction we only know that everything is different." (Grady Booch)
"To program is to model. To model is to understand." (Unknown)
"Virkeligheden er altings prøve" (Ukendt)

1. Introduktion til agentbaseret modellering 4
1.1. Mentale modeller 4
1.2. Computerbaserede modeller 6
1.3. Agentbaserede modeller 6
1.4. En tilnærmelse til naturligt sprog 9
1.5. Den decentraliserede tankegang – del 1 9
1.6. Reduktion af kompleksitet 10
1.7. En eksperimentel fremgangsmåde 11
1.8. Brug af modeller i undervisning og formidling 12
1.9. Formelle vs uformelle modeller 12
2. Introduktion til NetLogo 12
2.1. Smittemodel 13
2.1.1. Grænsefladen 13
2.1.2. Koden med procedurer 18
2.1.3. Sammenhæng mellem grænsefladen og koden 20
2.1.4. Flere procedurer i koden 21
2.1.5. At ændre i koden 24
2.2. Gærcellemodel 25
2.2.1. Interface-Tab 25
2.2.2. Info-Tab 27
2.2.3. Code-Tab 28
2.2.4. Intermezzo: turtles & patches 30
2.2.5. Setup-proceduren 31
2.2.6. Go-proceduren 33
2.2.7. Proceduren: flyt-til-en-tom-plads 35
2.2.8. Proceduren: lav-en-ny-turtle 37
2.2.9. Den decentraliserede tankegang – del 2 37
3. Ressourcer 38
Referencer 39

Forord

Modeller og modellering indgår i mange menneskelige aktiviteter. Ingeniører bygger modeller af broer, inden de går i gang med at bygge dem. Arkitekter bygger modeller af huse. Samfundsforskere udarbejder modeller, der kan bruges til at forudsige f. eks. arbejdsmarkedets udvikling under bestemte forudsætninger og med variationer på bestemte parametre. Litteraturforskere benytter modeller til at analysere tekster med, f.eks. modeller for en fortællings opbygning, der inkluderer begreber som spændingskurve m.m. eller modeller for lyrik, der benytter begreber som jamber og daktyler. Fysikere arbejder med modeller af naturens byggesten som atomer, lys og planeter. Biologer med modeller af cellemembraner og DNA-strenge. Sprogforskere kan arbejde med modeller for, hvordan sprog har udviklet sig over tid og geografi. Til grund for naturlige sprog ligger grammatikker, der afgør hvilke sætninger i et sprog, der er korrekte. Sådanne grammatikker kan opfattes som modeller for sprog.

Denne note giver en introduktion til modeller udarbejdet i værktøjet NetLogo. Som det senere vil fremgå tilhører de en klasse af modeller, der omtales som agentbaserede. Formålet er, at forklare hvordan sådanne modeller kan bruges til at undersøge fænomener, der har ens interesse. Målgruppen er således læsere, der ønsker at vide noget om, hvordan agentbaserede modeller kan bruges inden for deres eget fagområde. Noten er på ingen måde tænkt som udtømmende inden for emnet agentbaseret modellering eller værktøjet NetLogo, men derimod som et sted at starte, hvis man er nysgerrig på, hvad agentbaseret modellering er, og om det kan anvendes til at undersøge fænomener, man gerne vil undersøge og forstå bedre.

Læsevejledning: Noten er struktureret i tre dele. Første del fokuserer på agentbaseret modellering og beskriver karakteristika ved denne. Anden del fokuserer på NetLogo-værktøjet og gennemgår de basale og centrale dele af det. Tredje del indeholder referencer til yderligere ressourcer, hvis man vil vide mere om emnet.

Det anbefales, at man læser kapitel 1 i sin helhed inden man downloader og installerer NetLogo-værktøjet. Instruktioner om hvordan dette gøres er derfor placeret i starten af kapitel 2, hvor der også er indlagt en række øvelser.

Tak til: En stor tak til postdoc Line Have Musaeus, lektor emeritus Keld Nielsen (begge Aarhus Universitet), lektor Ole Eriksen (Erhvervsakademi Aarhus) og direktør Michael Caspersen (It-vest) for gode idéer, rettelser og kommentarer til noten. Også en stor tak til Jonas Ørbæk Hansen fra Silkeborg Gymnasium for at lade mig bruge hans smittespredningsmodel.

Palle Nowack, april 2023

1. Introduktion til agentbaseret modellering

Modeller indgår i rigtigt mange menneskelige aktiviteter. Det kan være fysiske modeller af huse, som dem arkitekter anvender, computerbaserede modeller af samfundsøkonomien, som dem finansministeriet anvender, eller mentale modeller, som alle mennesker implicit har om mange af de fænomener, som deres tanker er beskæftigede med. I dette kapitel introducerer vi agentbaseret modellering. Med udgangspunkt i begrebet mentale modeller beskriver vi fordele ved computerbaserede modeller generelt og agentbaserede modeller specifikt.

1.1. Mentale modeller

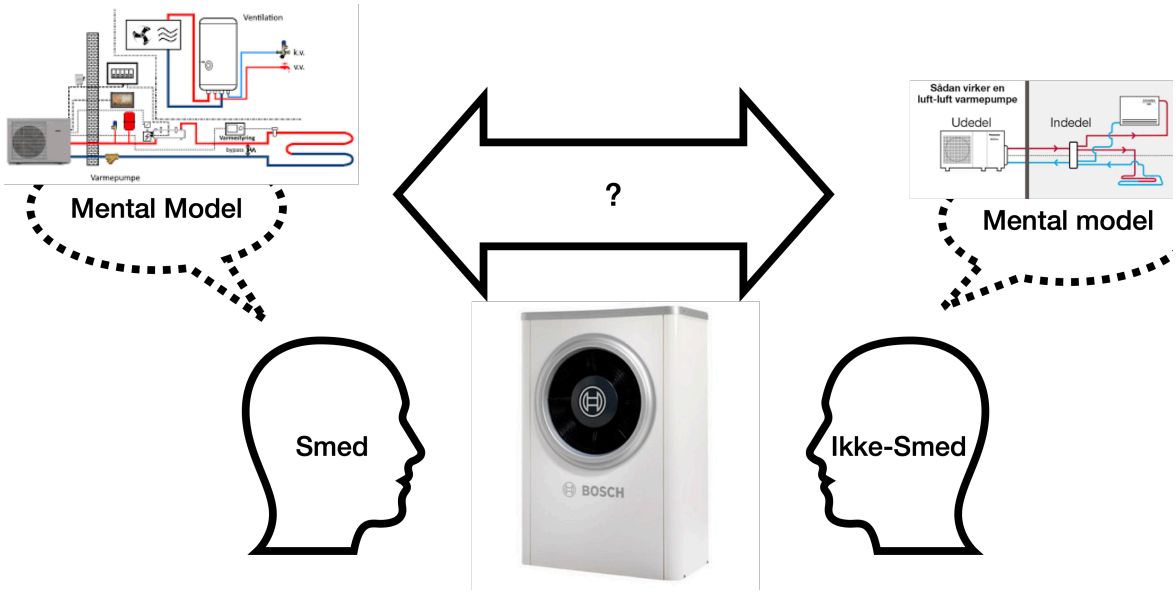
Mit hus opvarmes af en varmepumpe. Den varmer vand op og pumper det rundt i et lukket system, hvis vigtigste del er rørene i gulvvarmen og radiatorerne. På sin vej gennem rørene varmer vandet huset op, mens det selv afkøles og til sidst igen cirkulerer gennem varmepumpen.

De seneste måneder er det med jævne mellemrum sket, at trykket i systemet er faldet og jeg har måttet påfylde vand. I starten et par gange om måneden, men på det seneste 1-2 gange om ugen. Min egen konklusion var, at der måtte være en lækage i systemet. Det har der været før og årsagen var dengang, at en slange til gulvvarmen i et rum var utæt. Jeg havde derfor indstillet mig på en længere proces, hvor en specialist tømmer anlægget for vand og med et kompliceret apparat finder frem til den skjulte lækage. Derefter skifter man den utætte slange eller rør.

Ved det årlige eftersyn konstaterede den lokale vvs-mand samme fald i trykket. Hans konklusion var imidlertid helt anderledes. Modsat mig vidste han, at varmepumpen har et indbygget system - en såkaldt ekspansionsbeholder og en automatisk trykventil - der regulerer trykket. Ved overtryk strømmer vandet ind i ekspansionsbeholderen, der indeholder en gummimembran og den truende trykstigning afdæmpes. Trykventilen lukker luft ud af systemet, hvis der opstår luftlommer i det cirkulerende vand. Af erfaring vidste han at gummimembranen i ekspansionsholderen med tiden bliver slap, og at den automatiske trykventil også har en begrænset levetid. Begge dele medfører et trykfald i systemet. Men det vidste jeg ikke. Han skiftede de to dele, og problemet var løst med det samme. Vi undgik at skulle grave gulvet i stuen op igen. Et klassisk eksempel på at fagmanden har anderledes og bedre mentale modeller indenfor sit felt end en lægmand som jeg. Hans mentale model, som han brugte til at forstå, hvad der var galt med systemet, omfattede to ekstra elementer (ekspansionsbeholder og trykventil), som min meget primitive mentale model ikke indeholdt.

Vi bruger altså mentale modeller til dels at **forklare og forstå** vores omverden, dels til at **planlægge og tage beslutninger**. Vi benytter dem så meget, at vi normalt slet ikke er klar over det. Men når vi oplever en uoverenstemmelse mellem det vi oplever og vores evne til at "forstå" det, så bliver vi pludseligt opmærksom på vores mentale model og behovet for at udbygge eller ændre den.

Når vi lærer noget nyt tilføjer vi det til vores mentale modeller, eller vi ændrer i de eksisterende mentale modeller. Fordi mennesker ikke er ens, har vi forskellige mentale modeller, se figur 1. Der er typisk store overlap, men altså også forskelligheder. Det kan der være mange grunde til: f.eks. erfaring, specialistuddannelser eller etiske synspunkter og politiske holdninger.



Figur 1: Mennesker har forskellige mentale modeller om de samme fænomener.

Noget af det vanskelige ved at arbejde med mentale modeller er, at de kan være svære at kommunikere til andre. To mennesker kan godt have en fornemmelse af, at de har samme mentale model af nogle fænomener og deres sammenhænge, men det kan være særdeles vanskeligt at afgøre, om det egentligt er tilfældet. Man kan bruge meget lang tid og mange ord på at finde ud af, om man deler den samme mentale model om et fænomen.

Derfor er det tit en givende proces at forsøge at få sin mentale model udtrykt i et andet fysisk materiale end ord på papir eller ord i luften. I vvs-eksemplet kan man benytte sig af diagrammer, der viser varmepumpens forskellige dele og deres sammenhænge. Man kan pege og forklare via en sådan fysisk model. En ulempe ved fysiske modeller kan være, at de kan være kostbare at lave, og det kan tage lang tid at lave dem. En anden ulempe er, at de typisk er meget statiske. Papir er meget statisk. De kan vise elementer og strukturer (beholdere og rør), men sjældent dynamiske processer (vandets flow og skiftende temperatur, det skiftende tryk i systemet), elementers interaktion (vandet og gummimembranen) og udvikling over tid (gummimembranen der bliver slap).

Øvelse 1: Mental model af en kaffemaskine.

- Overvej hvordan en almindelig filterkaffemaskine virker.
- Skitser de forskellige dele og deres sammenhænge.
- Overvej hvad der kan være problemet i de følgende scenarier:
  - Maskinen laver kold kaffe.
  - Maskinen producerer ikke kaffe, men kun varmt vand.
  - Maskinen producerer ingenting

1.2. Computerbaserede modeller

Hvis man vil beskrive et fænomen, som omfatter dynamiske processer, elementer der interagerer og udvikling over tid, er computersimulationer et oplagt redskab at benytte. De har mange fordele: de kan være **visuelle**, **interaktive**, **dynamiske**, relativt **hurtige** at lave, relativt **billige**, særdeles **omfangsrige** (dvs. med mange data og/eller med mange beregninger) og de kan **parameteriseres**. Sidstnævnte betyder, at man i modellen indbygger et antal "håndtag", som man kan skrue på. I en finansmodel kan det f.eks. være det forventede antal af ledige, ældre og børn, i en bromodel kan det være den forventede styrke af betonen og strømforholdene, og i en smittemodel kan det være antallet af dage, en person skal være smittet, førend vedkommende bliver immun. Disse parametre kan man så sætte til forskellige værdier og se hvordan modellen reagerer: stiger skattetrykket? eroderer broen? og bliver alle immune med tiden eller uddør befolkningen? Udover at man manuelt kan sætte udvalgte parametre til faste værdier, kan man også indbygge **tilfældigheder**. Dvs. at man for udvalgte parametre ikke sætter en fast værdi, men i stedet beder computeren generere en tilfældig værdi f.eks. i et interval mellem 1,5 og 2,5 for antal børn født pr. kvinde. Igen kan man køre sin model og se hvordan den reagerer på disse tilfældige input. Man kan udføre sin computersimulation (dvs. "køre sin model" - vi bruger begge vendinger i flæng i denne note) 1000 gange med varierende og tilfældige parametre og se på forskelle og ligheder i både processerne og slutresultatet.

1.3. Agentbaserede modeller

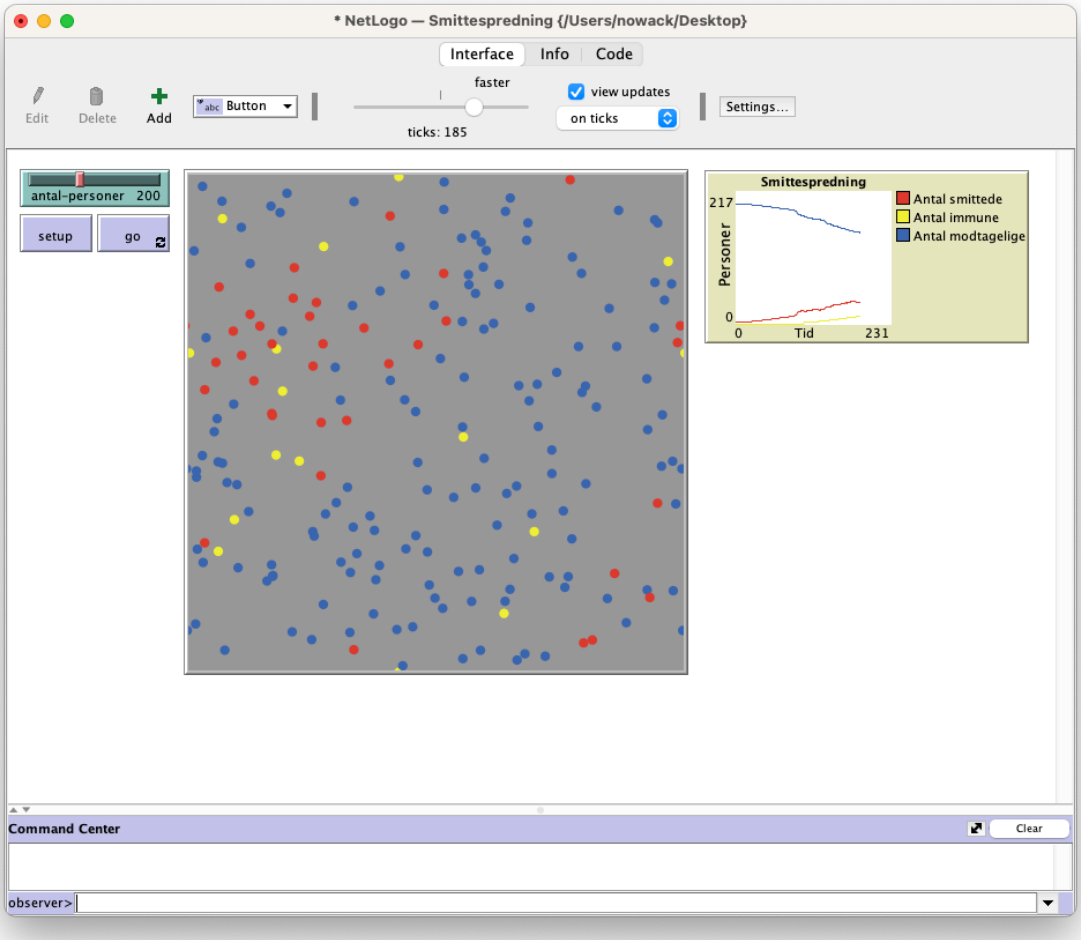
Denne note introducerer en bestemt type af computerbaserede modeller, der kaldes agentbaserede modeller. I en agentbaseret model kigger man på de fænomener, som man vil undersøge, som en samling agenter, der interagerer med hinanden. En agent er en selvstændig enhed, der har sin egen tilstand og sin egen adfærd. En agent kan f.eks. være en person, en celle, et atom osv. Man vælger altså at forstå sit problemområde som bestående af en række agenter, og man bygger sin model op som en samling af sådanne agenter. Det er vigtigt at understrege, at det er et **perspektiv** man lægger ned over virkeligheden. Det er et valg, man foretager. Man kunne også have valgt at bygge sin model af legoklodser, som grafiske symboler på et stykke papir eller matematiske differentialligninger, men vi vil i denne note demonstrere de mange fordele, der er ved at bygge sin model op af agenter. Til det formål skal vi bruge et eksempel.

11. marts 2020 afholdte statsminister Mette Frederiksen ekstraordinært pressemøde, hvor hun indledte med ordene: "Situationen omkring corona udvikler sig, og den er desværre meget alvorlig. Smittetallene er høje. Mere end 22.000 er konstaterede smittede den seneste uge mod omkring 15.000 ugen før."[1]. I de kommende måneder og år blev ord som smitte og smittespredning allemandseje, og de kom til at fylde rigtigt meget i mange mennesker liv. Vi udviklede derfor nok alle en mental model af fænomenet "smittespredning" og vi brugte den hver især til at prøve at forstå, hvad der skete med os og samfundet. Vi brugte den også til at prøve at opføre os på en måde, der var i overensstemmelse med vores holdning til

[1] https://www.stm.dk/presse/pressemoedearkiv/pressemoede-den-16-december-2020/

smittespredningen. For de fleste involverede det brug af maske, isolering i "bobler", modtagelse af vacciner, o.s.v.

I denne note vil vi anvende en model af smittespredning som vores gennemgående eksempel. Modellen er illustreret i figur 2, der viser et snapshot på et givent tidspunkt. Man skal altså forestille sig, at modellen er dynamisk og har kørt et stykke tid. I den tid har personerne (de farvede cirkler, der repræsenterer mennesker) bevæget sig rundt og nogle af dem har skiftet farve. I denne model har vi altså valgt, at agenterne repræsenterer personer i en population.



Figur 2: En agent-baseret model af en population udsat for smitte.

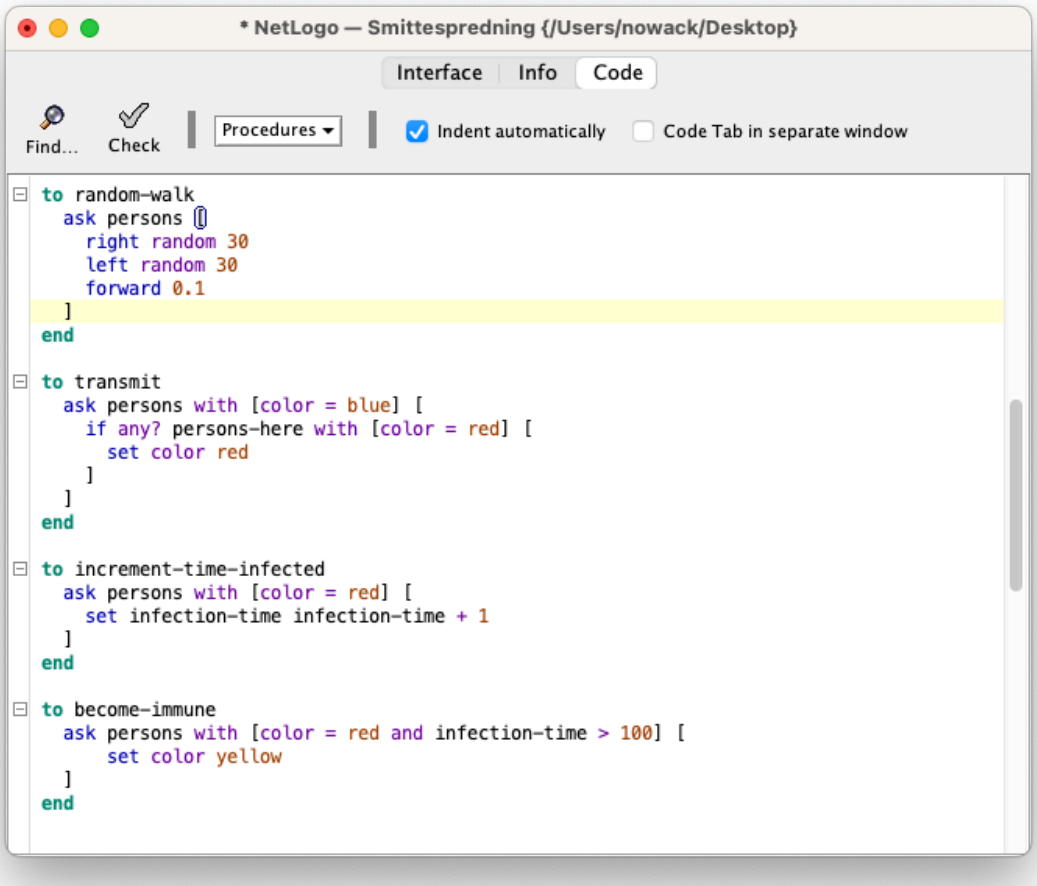
Agenterne i denne model kan have tre forskellige tilstande (symboliseret ved deres farve): de kan være modtagelige for smitte (de blå), de kan være smittede (de røde) eller de kan have været smittede, men er nu blevet immune (de gule). Alle agenter følger de samme 4 simple regler for adfærd:

- Alle agenter bevæger sig tilfældigt rundt.
- Hvis en usmittet agent møder en smittet agent, bliver førstnævnte smittet.
- En smittet agent husker på, hvor mange dage den har været smittet
- Hvis en agent har været smittet i en periode på 100 dage, så bliver vedkommende immun.

Hovedparten af den kode, der implementerer denne model, er vist i figur 3. Det er ikke vigtigt, at du forstår alle detaljerne i denne kode lige nu, blot du kan følge de overordnede linjer.

Koden består af fire kodeblokke. Hver kodeblok fastlægger en regel for adfærd, som alle agenter skal overholde:

- "random-walk" beder alle agenterne ("persons") dreje sig i en tilfældig vinkel mellem 0 og 30 grader til højre, derpå dreje sig i en tilfældig vinkel mellem 0 og 30 grader til venstre og endelig gå et lille skridt fremad. Resultatet er at personerne går tilfældigt rundt i mellem hinanden.
- "transmit" beder alle de blå (usmittede) agenter finde ud af, om de står i nærheden af nogle (smittede) røde agenter. Hvis det er tilfældet, så bliver de selv røde (smittede). Hvis ikke, sker der ingenting med dem.
- "increment-time-infected" beder alle de røde agenter huske på, at de nu har været smittet en dag længere.
- "to-become-immune" beder alle agenter, der er smittede (røde) og har været smittede i mere end 100 dage, om at farve sig selv gule for at symbolisere, at de nu er blevet immune.



Figur 3: De centrale dele af koden bag den agentbaserede smittemodel.

Sammenfattende er det altså vigtigt at forstå, at der er tæt sammenhæng mellem den adfærd, som kan observeres gennem den grafiske brugergrænseflade i figur 2 og så den underliggende kode, der er illustreret i figur 3.

1.4. En tilnærmelse til naturligt sprog

En særlig fordel ved agentbaserede modeller er, at modellerne er udtrykt i noget, der minder om naturligt sprog, modsat f.eks. matematiske udtryk. Figur 4 viser koden for den adfærdsregel, der bestemmer om en person bliver immun.

```
to become-immune
  ask persons with [color = red and infection-time > 100] [
      set color yellow
  ]
end
```

Figur 4: Koden der afgør om en person er blevet immun.

Koden betyder, at vi erklærer en procedure, der har navnet "become-immune". En procedure er en samling kommandoer, som vi giver et navn. Når man fra et andet sted i koden kalder proceduren ved at skrive procedurens navn udføres alle procedurens kommandoer. Vi uddyber procedurebegrebet i afsnit 2.1. Når vi kalder proceduren "become-immune", så udfører computeren alle instruktionerne i proceduren og beder alle personer, der er røde og som har været smittede i mere end 100 dage, om at sætte deres farve til gul for at signalere, at de er blevet immune.

NetLogos programmeringsprog er et eksempel på et imperativt programmeringssprog, der fungerer på den måde, at den udfører kommandoer (imperativer), der aflæser og opdaterer systemets tilstand. I vores tilfælde aflæser vi agenternes tilstand, dvs. om agenterne er røde og smittede i mere end 100 dage. Hvis det er tilfældet opdaterer vi deres tilstand, dvs. ændrer deres farve til gul.

1.5. Den decentraliserede tankegang – del 1

Agentbaserede modeller anvender et lokalt perspektiv fremfor et globalt perspektiv. I stedet for at skrive programkoden fra et enkelt centralt top-down perspektiv, hvor alle systemets mulige tilstande og kombinationer af tilstande skal være forudset og eksplicit håndteret, så fordeler man sin programkode på små, mindre enheder (agenter og procedurer), hvor hver agent har ansvar for at håndtere sin egen adfærd. Dette kaldes **den decentraliserede tankegang** og den er helt essentiel for agentbaseret modellering.
Når man f.eks. oplever "sort sol" fænomenet, hvor store stæreflokke samles og flyver rundt i luften, kan det modelleres ved, at den enkelte stær følger tre simple adfærdsregler:

- Flyv i samme retning og med samme hastighed som mine naboer.
- Flyv ind mod flokkens tyngdepunkt.
- Flyv ikke for tæt på naboerne.

Når de enkelte stære følger disse tre simple regler, opstår en tilsyneladende kollektiv

bevægelse, hvor stærenes bevægelser er koordinerede, men uden central kontrol. Der sidder ikke en chef-stær i flokken og fortæller, hvordan de andre stære skal flyve. Dette kaldes også for emergent adfærd: helhedsadfærden opstår som en konsekvens af de individuelle deles adfærd.

Omsat til modeller og kode, så giver en sådan organisering en meget simpel struktur af koden, da koden hele tiden kun skal tage stilling til lokale fænomener. I vores tilfælde med smittemodellen: Er der nogen smittede i nærheden af mig? Bliver jeg selv smittet nu? Hvad er min farve? Hvad er farven på de andre som er tæt på mig? Hvor længe har jeg været smittet? Flyt mig selv lidt rundt. Det emergerende resultat kan i dette tilfælde aflæses i kurverne i plottet over raske, smittede og immune personer yderst til højre i figur 2.

Konceptet kan forekomme lidt abstrakt på nuværende tidspunkt, men vi vender tilbage til det til sidst i afsnit 2.

1.6. Reduktion af kompleksitet

Når man laver en model som den førnævnte smittemodel, foretager man en række **abstraktioner**. I smittemodellens tilfælde vælger vi f.eks. at modellere personer som agenter. Vi vælger, at de alle har samme adfærd: de bevæger sig tilfældigt rundt, en smittet smitter altid en usmittet person, hvis de er i nærheden af hinanden, og efter 100 dage bliver en smittet person immun. Det er jo en klar simplificering i forhold til virkeligheden. Man kunne have valgt at designe sin model, så der var områder med høje og lave befolkningstætheder, som f.eks. by- og landområder. I stedet for at alle bevæger sig tilfældigt rundt i et ensartet område, kunne man have introduceret boligkvarterer, butikscentre, daginstitutioner, naturområder, osv. Man kunne have lavet ruter, som de fleste agenter vælger at bevæge sig indenfor. Endvidere kunne man have indbygget noget sandsynlighed omkring selve smitten, således at en smittet person ikke altid smitter en usmittet person, men f.eks. lade det ske i 30% eller 60% af alle møder. Man kunne have ladet det være tilfældigt om en person blev immun efter 75, 100 eller 125 dage. Man kunne have valgt at en vis procentdel af befolkningen aldrig kunne blive smittet eller aldrig blive immune. Man kunne have introduceret forskellige typer af agenter: nogle der er flittige med håndsprit og mundbind, nogle der ignorerer disse ting, nogle der er flittige til selvisolering, nogle der ikke lader sig isolere.

En model er således altid udtryk for et valg: man vælger et perspektiv (hvad skal være vores agenter?), man vælger agenternes adfærd og effekten af denne adfærd. Man vælger hvilke dele af virkeligheden, der skal være en del af modellen og hvilke dele man ignorerer og ser bort fra. Andre valg vil give anderledes resultater.

Det er vigtigt at huske, at en model ikke er virkeligheden. Alle modeller er tilnærmelser og dermed de-facto "forkerte",[2] men nogle modeller kan være meget nyttige. Det skyldes, at modeller er udtryk for abstraktion: når vi laver en model, lægger vi vægt på visse dele/aspekter og ignorerer andre dele. Det betyder også, at en model altid udtrykker en bestemt synsvinkel på de betragtede fænomener. En del af denne synsvinkel er ofte modellens størrelsesforhold (ofte kaldet modellens granularitet), dvs. hver gang man konstruerer en

[2] Som man siger i militæret: hvis der er uoverenstemmelser mellem kort og terræn, så stol på terrænet.

model, skal man tage stilling til, hvad de mindste bestanddele af modellen skal være.

Sammenfattende kan man sige, at når vi bygger en model, så fravælger vi typisk en detaljerigdom til fordel for præcision og et bestemt fokus. Vores modeller er fattigere på detaljer end virkeligheden (det vi forsøger at beskrive). Til gengæld bliver det nemmere at holde fokus på bestemte aspekter ved den virkelighed. Modellen **reducerer kompleksiteten** i vores forståelse af virkeligheden.

1.7. En eksperimentel fremgangsmåde

At arbejde med modeller er i udgangspunktet en eksperimentel arbejdsproces. Dette gælder både i brugen af og i udviklingen af modeller.

Når man bruger modeller har man ofte mulighed for at variere værdien af parametre igennem modellens grafiske brugergrænseflade. I vores simple smittemodel kan man variere populationens størrelse via modellens indbyggede skydeknap. Man kan sætte skydeknappen til forskellige værdier og se hvordan det påvirker, hvordan modellen kører og resultatet af processen over tid. I vores model kan man f.eks. aflæse, hvordan kurverne for antallet af usmittede, smittede og immune ændrer sig, når vi ændrer på populationens størrelse.

Når man udvikler modeller, anbefaler vi, at man starter simpelt: lav den mindst mulige model, der kan afvikles og vise et resultat. Dvs. start med få agenter, en relativt lille verden, samt en eller to simple regler for adfærd. Når denne model er etableret og kan køre uden at melde fejl, så kan man gradvist tilføje flere og flere agenter, øge omfanget af verden, samt øge antallet og kompleksiteten af adfærdsregler.

I praksis vil man i arbejdet med en model typisk veksle i mellem at ændre på værdierne af parametre gennem brugergrænsefladen og ændre i den underliggende kode i modellen. En vigtig designbeslutning ved udviklingen af modeller er netop at besluttet, hvilke parametre der skal kunne varieres på gennem brugergrænsefladen.

Sammenfattende bruger vi denne eksperimentelle fremgangsmåde til at **reducere usikkerhed** og fremskaffe mere viden. Hvis vi ikke kender fænomenernes sammenhæng i problemområdet, så kan man prøve at modellere det på ene eller anden måde, og se hvilket resultat, der stemmer bedst overens med den observerbare virkelighed eller teorier på området.

En sådan eksperimentel brug af modeller er ikke begrænset til agentbaserede modeller specifikt eller computerbaserede modeller generelt. Hvis vi f.eks. i en gruppe sidder med en fysisk plastikmodel af et molekyle, så kan vi se de forskellige bindinger og atomer, vi kan fjerne nogle og tilføje andre. På den måde bliver det klart for alle, hvad det enkelte medlem af gruppen mener, når vedkommende forklarer og fortæller. Vi kan diskutere, om det er korrekt eller hensigtsmæssigt, i forhold til det vi ønsker at opnå eller beskrive, men muligheden for misforståelser er klart mindre, end hvis vi bare sidder og taler ud i den blå luft og fægter lidt med armene.

På samme måde anvender f.eks. arkitekter fysiske modeller i deres samarbejde med kunder og ingeniører. Ud fra en model af en ny bygning kan man diskutere om rummene og forbindelserne imellem disse er hensigtsmæssige og funktionelle, ligesom man kan diskutere æstetiske aspekter ved dele og helheden. Denne måde at reducere usikkerhed på, er klart billigere end at lave den virkelige bygning, rive ned og bygge op.

1.8. Brug af modeller i undervisning og formidling

Modeller og modelleringsprocessen er særdeles interessante i læringsmæssige sammenhænge. Man kan bruge modeller til at tænke med.

I det ene ekstrem kan underviseren udlevere eller henvise til eksisterende (veletablerede og velafprøvede) modeller som f.eks. Bohrs atommodel. Den studerende kan lære af de udleverede modeller, eksperimentere med dem, og danne sine egne synspunkter på nogle fænomener der repræsenteres af modellerne. De udleverede modeller kan udstille forskellige aspekter af de samme fænomener og de kan udstille forskellige (og sågar modstridende) tolkninger af de samme fænomener.

I det andet ekstrem kan underviseren bede de studerende om at bygge modeller, der udstiller elevernes forståelse af et fænomen på et givent tidspunkt. Dette giver underviseren indsigt i, hvordan de studerende tænker, og kan dermed benyttes til at styre undervisningen i en bestemt retning, hvis det f.eks. viser sig, at mange studerende har samme misforståelser af et bestemt fænomen.

Med vores smittemodeleksempel vil en lærer f.eks. meget hurtigt kunne få en dialog med sine studerende om manglerne ved modellen i forhold til virkeligheden, og grupper af studerende kan få til opgave at forbedre modellerne ud fra netop deres fokuspunkt. Modeller er et glimrende udgangspunkt for fagligt fokuseret kommunikation.

1.9. Formelle vs uformelle modeller

Modeller kan være mere eller mindre formelle. Efter fodboldkampen kan målmanden over en velfortjent pølsemix i cafeteriet forklare medspillere, hvordan de burde have dækket op ved modstanderens tredje scoring. Salt- og peberbøsser kan repræsentere forsvarsspillere, to pomfritter modstanderens angribere. Inden kampen havde træneren måske brugt et whiteboard med magneter til at repræsentere spillerne og en tusch til at demonstrere bold- og løberetninger, da dagens taktik skulle forklares.

I den mere formelle ende af spektret benytter f.eks. finansministeriet sig af modeller til at regne på samfundsøkonomiens udvikling under forskellige forudsætninger. Disse modeller kan have hundredevis af parametre man kan justere på og dermed millioner af kombinationsmuligheder. Endelig kan et sæt matematiske differentialligninger benyttes til at beskrive hvordan en bølge udbreder sig i vand eller hvordan smitte af f.eks. Covid kan sprede sig i en population.

Vores eksempel med smittemodellen er klart en formel, omend simpel, model.

2. Introduktion til NetLogo

Eksemplet i foregående afsnit er — som allerede nævnt — lavet i det agentbaserede værktøj, der hedder NetLogo. Der findes mange andre ABM-værktøjer,[3] men i denne note benytter vi udelukkende NetLogo, hvilket primært skyldes dets "low threshold, high ceiling" filosofi. Dvs. at det er relativt nemt at lære, samtidigt med at det er avanceret nok til at kunne lave

[3] ABM: AgentBased Modeling

endog meget komplekse modeller. I dette kapitel kigger vi nærmere på NetLogo-værktøjet, dets bestanddele og hvordan de forskellige dele hænger sammen. Vi starter med at kigge på smittemodellen fra det forrige kapitel, og bagefter kigger vi på et andet eksempel, nemlig en model af gærcellers vækst.

Øvelse 2: Download & Installer.

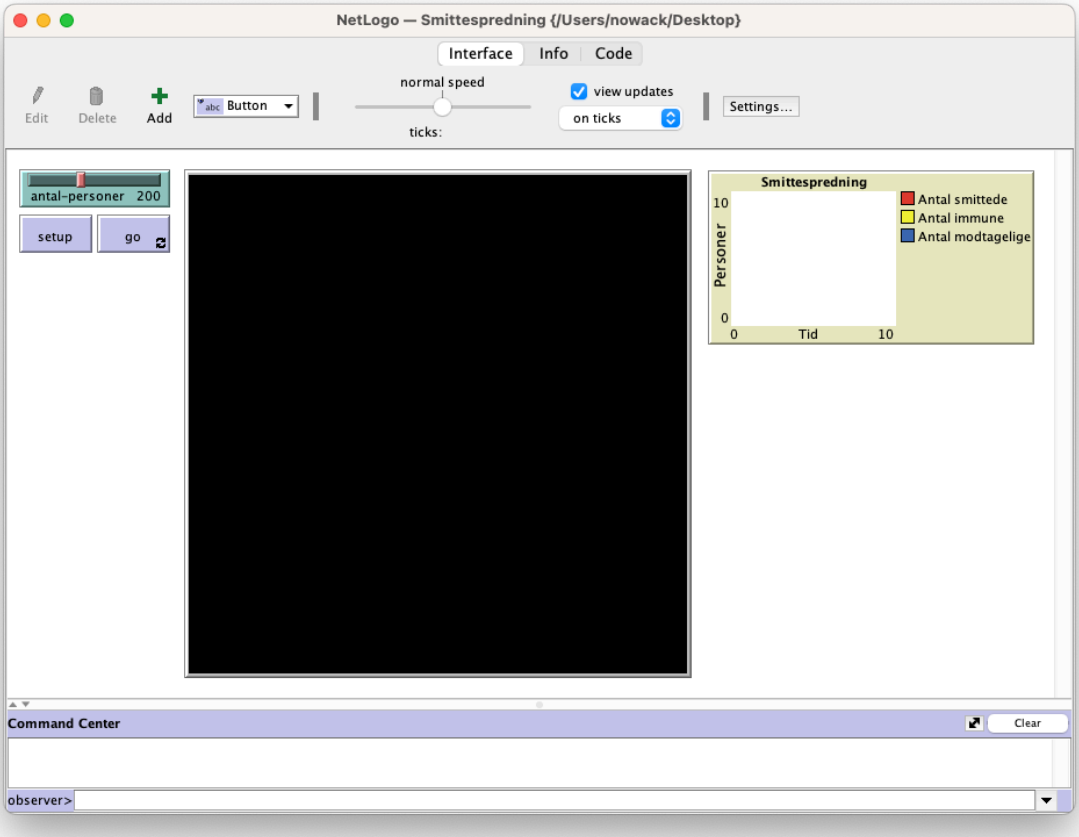
- Download og installer NetLogo værktøjet fra: https://ccl.northwestern.edu/netlogo/download.shtml
- Download de to eksempelmodeller "Smittespredning.nlogo" og "Gærcellevækst.nlogo" fra ??? (hvor skal de placeres?)
- Når du har downloadet og installeret NetLogo værktøjet, åbner du applikationen og inde fra denne, åbner du så den første model "Smittespredning" via File-menuens Open-punkt. Alternativt kan du dobbeltklikke på "Smittespredning" modelfilen.

2.1. Smittemodel

I det følgende bruger vi smittemodellen til at introducere NetLogos to centrale faneblade: grænsefladen, koden og sammenhængen imellem disse to. Endvidere beskriver vi det centrale begreb om procedurer i NetLogo-koden.

2.1.1. Grænsefladen

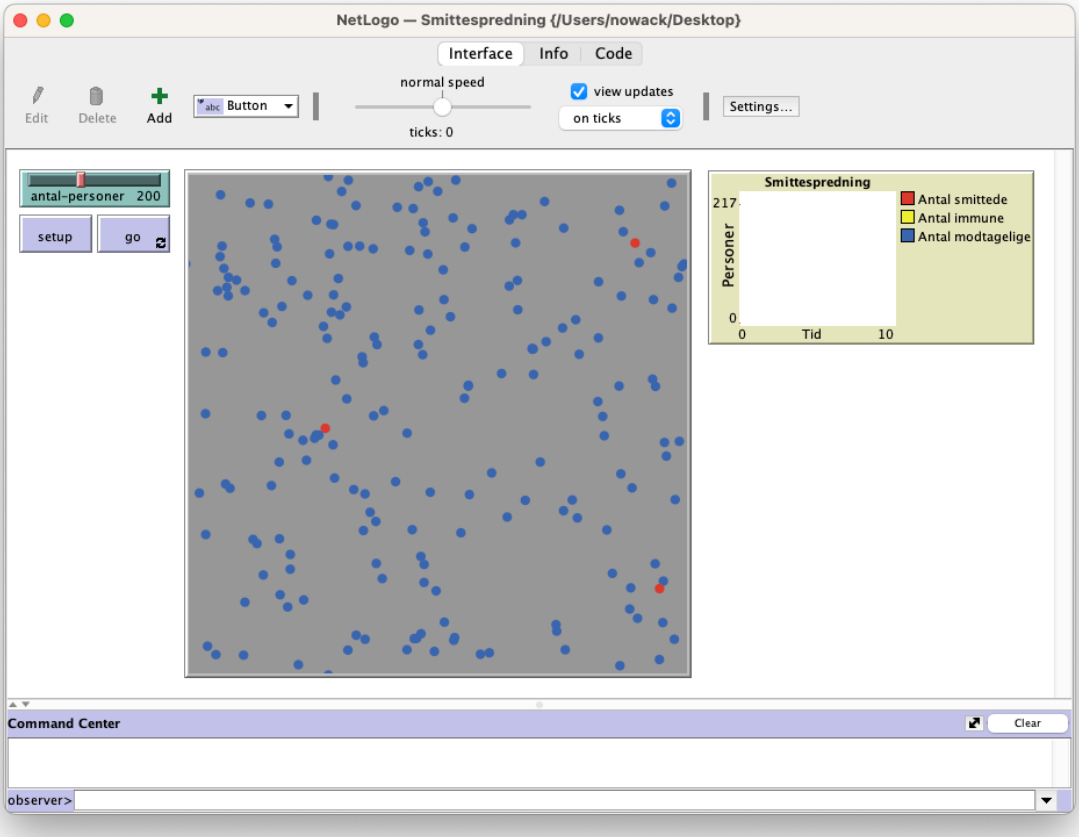
I det følgende kigger vi nærmere på smitteeksemplet fra det foregående afsnit. Når man åbner modellen i NetLogo, får man et vindue, der er vist i figur 5.



Figur 5: NetLogo-værktøjet når man har indlæst Smittespredningsmodellen.

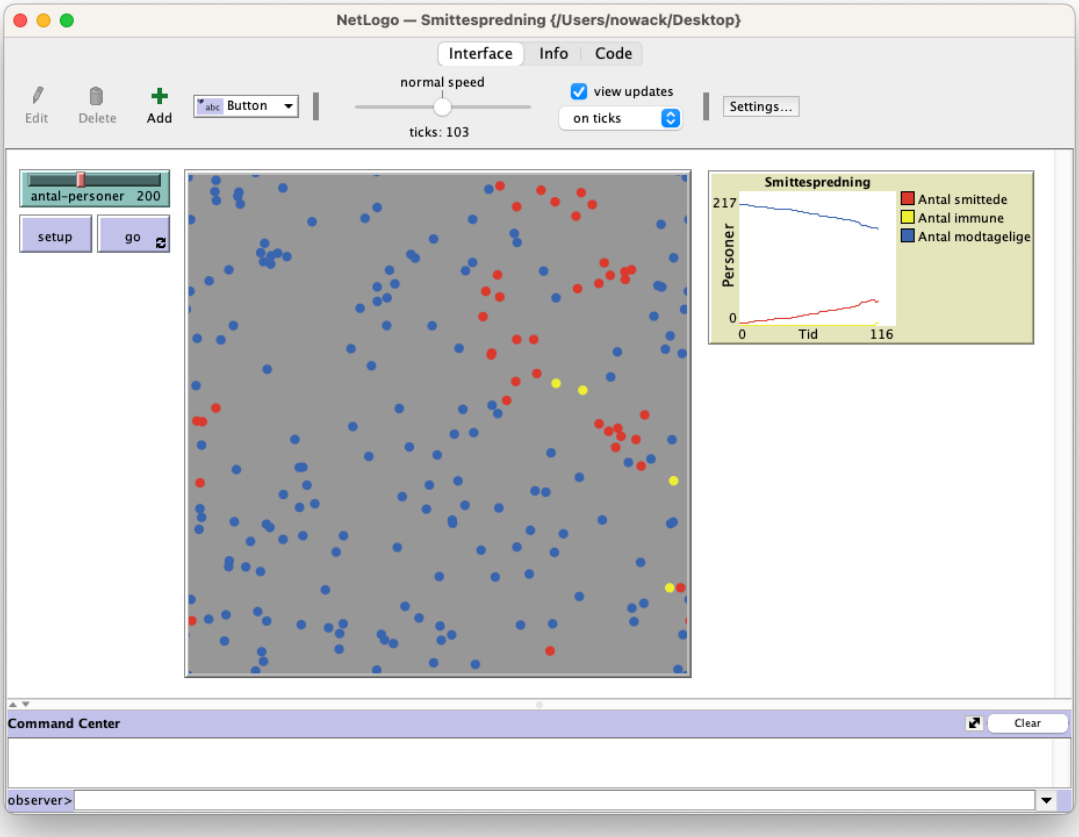
Vinduet er opdelt i tre vandrette bjælker. Den øverste grå bjælke indeholder i toppen tre knapper, der skifter mellem de tre faneblade: Interface, Info og Code. Ved at trykke på knapperne skifter man mellem fanebladene. Ved opstart er man på fanebladet Interface, som i figur 5. Bjælken indeholder andre elementer, som vi vender tilbage til. Den midterste bjælke indeholder fra venstre: en slider, der bestemmer værdien af variablen "antal-personer", herunder to lilla knapper ("setup" og "go"), så følger et stort sort (tomt) felt, og til højre er der en tom graf, der kan vise antal smittede, antal immune og antal modtagelige i modellen over tid. Nederst er der et "Command Center", hvor man kan indtaste kommandoer, men det skal du blot ignorere indtil videre.

Hvis man trykker på knappen "setup", ændrer visningen sig til det, man kan se på figur 6. Det sorte felt er nu ændret til en grå baggrund, hvorpå der er fordelt en mængde blå og røde prikker. De blå prikker repræsenterer personer, der kan blive smittet, og de røde repræsenterer personer, der allerede er smittede. Som det fremgår er tre personer smittede her ved opstart af modellen.



Figur 6: NetLogo vinduet, når man har trykket på “setup” knappen.

Bemærk også, at i den øverste bjælke, lige under den slider, der kontrollerer modellens hastighed, står der “ticks: 0”. Ticks er NetLogos indbyggede tidsenhed, der starter ved 0 og tælles op med 1, hver gang modellen har udført et skridt af simuleringen (dvs. udførslen af modellen). Nu prøver vi at trykke på knappen “go”, se figur 7. Nu sker der noget! Vi oplever at alle personerne går rundt i mellem hinanden. Når de blå personer kommer i nærheden af de røde, så kan vi se, at de blå skifter farve til rød. Med andre ord: de smittede personer smitter de usmittede (modtagelige) personer.

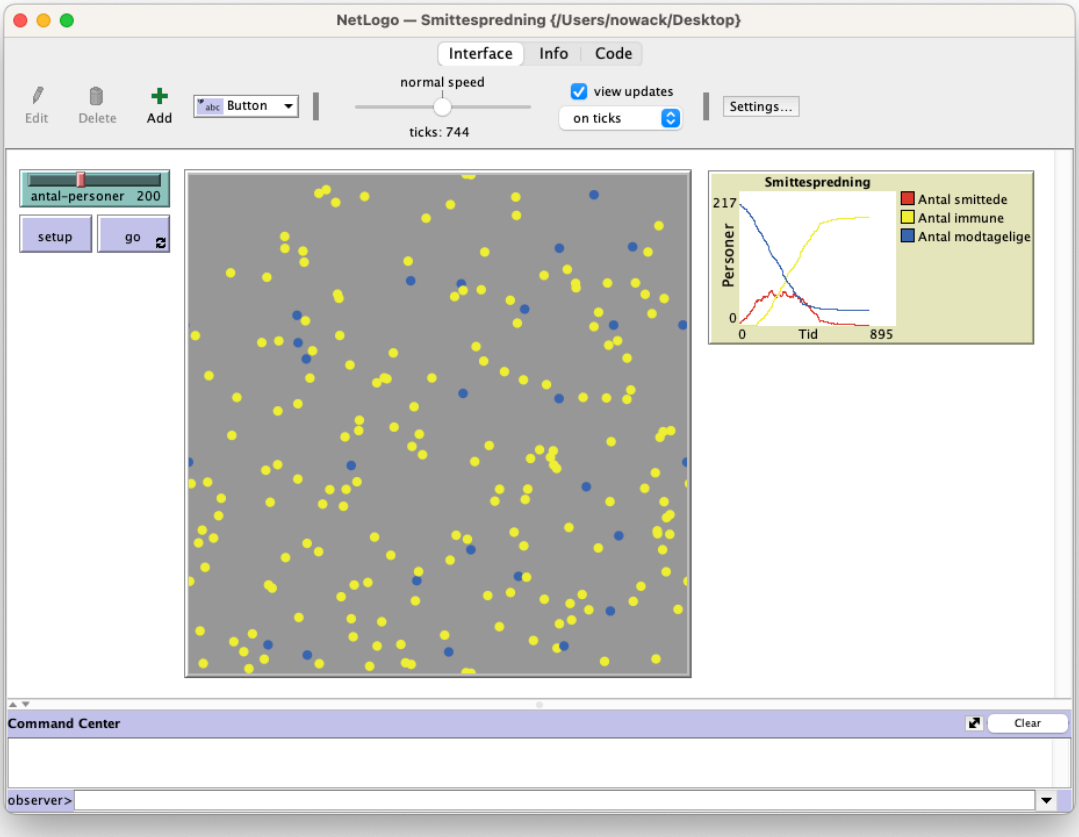


Figur 7: NetLogo vinduet efter ca. 100 ticks, når man har trykket på “go” knappen.

Når der er gået noget tid (ca. 100 ticks), så oplever vi også, at nogle af de røde personer skifter farve til gul. Det repræsenterer, at de smittede personer er blevet immune. De gule personer kan herefter ikke længere smittes. Læg også mærke til graferne til højre. De viser udviklingen i antallet af de tre typer personer, vi har i vores model: smittede, immune, og modtagelige.

Man kan til enhver tid stoppe udførelsen af modellen ved at trykke på “go” knappen igen. Så sættes modellen på pause, og den kan sættes i gang igen ved at trykke på “go” knappen endnu engang. Man kan også ændre på hastigheden af modellen ved at trække i slideren i midten af den øverste bjælke. Man oplever at modellen opdateres langsommere (træk mod venstre) eller hurtigere (træk mod højre), og man kan også observere at “tick” tælleren tæller i et tilsvarende anderledes tempo.

Efter godt 700 ticks ser modellen ud som illustreret i figur 8. Bemærk, at der nu ikke er flere røde (smittede) personer, hovedparten af befolkningen er blevet immune (de gule), og der er ganske få personer, der ikke har været smittede (de blå).



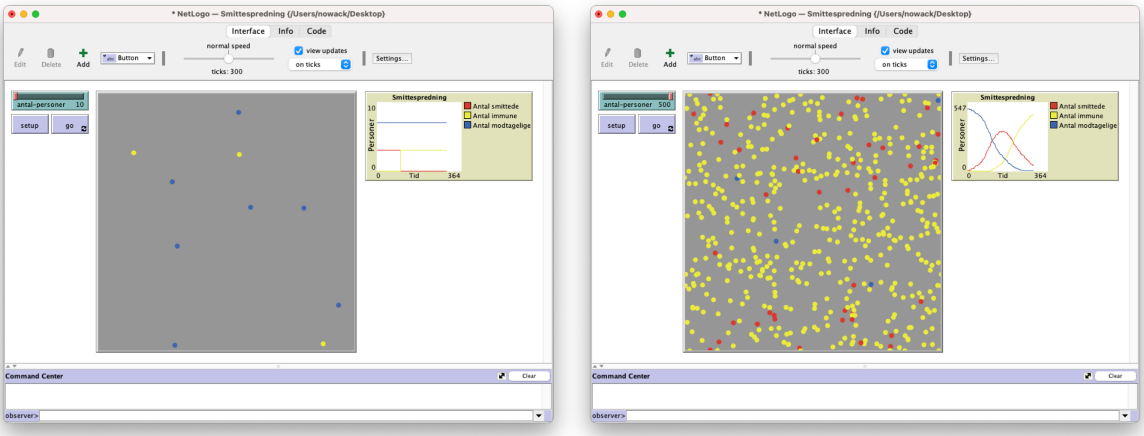
Figur 8: NetLogo vinduet efter godt 700 ticks.

Læg også mærke til graferne i højre side, hvor man kan følge udviklingen af smitten. Bemærk, at både den blå og gule kurve flader ud med tiden, samt at den røde kurve først stiger kraftigt og herefter falder kraftigt, til den flader ud.

Øvelse 3: Setup & Go knapperne

- Åben smittespredningsmodellen i NetLogo.
- Eksperimenter med at trykke på “setup” og “go” i forskellige rækkefølger og observer, hvad der sker.
- Prøv også at ændre hastigheden på modellen ved at trække i slideren i den øverste grå bjælke.

Hvis man vil se, hvordan størrelsen af populationen påvirker modellens adfærd og resultat, så kan man ændre den ved at trække i den grønne slider øverst til venstre. Dette skal gøres, inden man trykker på “setup”-knappen. I figur 9 ses hvordan modellen ser ud efter 300 ticks ved at indstille slideren på de to yderpunkter (dvs. henholdsvis 10 og 500). Bemærk specielt udseendet af graferne i de to udførsler.



Figur 9: NetLogo vinduet efter 300 ticks ved en population på 10 (venstre) og 500 (højre).

Øvelse 4: Ændre værdien af en variabel

- Åben smittespredningsmodellen i NetLogo.
- Sæt slideren, der bestemmer værdien af variablen “antal-personer” til en værdi, og tryk herefter på “setup” og “go” knapperne. Observer, hvad der sker. Gentag med forskellige værdier.
- Prøv også at sætte slideren til en ny værdi, mens modellen kører. Sker der noget?

En indskudt bemærkning: denne brug af sliders og knapper demonstrerer et generelt design-mønster, når man udvikler modeller i NetLogo. De kontrolenheder (som i dette tilfælde er den grønne slider, der kontroller størrelsen af populationen) der skal bruges, inden man trykker på “setup” og “go” knapperne, placeres typisk ovenfor eller til venstre for disse. Man kan også have sliders, knapper eller andre kontroller, som det giver mening at anvende, når modellen er sat i gang, og dem placerer man typisk under eller efter “setup” og “go” knapperne. På den måde udnytter man vores naturlige tilbøjelighed til at læse og forstå ting fra venstre og mod højre, oppefra og nedefter.

Sammenfattende har vi nu været igennem en første introduktion til grænsefladen af NetLogo værktøjet. Vi har set hvordan den består af flere faneblade, knapper og andet til at kontrollere udførslen af modellen, et grafisk felt, der illustrerer agenterne og deres adfærd i modellen, samt muligheden for at lave plots af udvalgte data.

2.1.2. Koden med procedurer

Koden, der implementerer vores smittemodel, er vist i figur 10. Bemærk, at vi nu har skiftet til fanebladet “Code” (se øverst i den grå bjælke). De mange linjer kode kan virke uoverskueligt ved første øjekast, men vi tager det lidt ad gangen.

Når man skal læse kode, er det sjældent en god idé at starte ovenfra, læse en linje ad gangen og forsøge at forstå den linje, inden man fortsætter. Hvis man forfølger den strategi, vil man ofte opleve, at man ret hurtigt sidder fast i sine bestræbelser på at forstå, hvad et program gør. Koden er jo netop skrevet på computerens betingelser, for at den kan udføre koden korrekt. Det indebærer f.eks. ofte at bestemte dele af koden skal komme i en bestemt rækkefølge. Vi mennesker forstår derimod ting lidt anderledes, og forsøger f. eks. ofte at associere os frem til overordnede sammenhænge, som vi så igen bruger til at fortolke

enkeltelementer i disse sammenhænge.


```
breed [persons person]
persons-own [person-infection-time]

globals [
  expected-infection-time ;; variabel der indeholder smitteperiodens forventede varighed
]

to setup
  clear-all
  setup-patches
  setup-persons
  set expected-infection-time 100  ;; Sundhedsstyrelsens aktuelle tal for smitteperiodens varighed
  reset-ticks
end

to go
  random-walk
  transmit
  increment-time-infected
  become-immune
  tick
end

to setup-patches
  ask patches [set pcolor grey]
end

to setup-persons
  create-persons antal-personer [
    set shape "dot"                  ;; ikon for personer
    set color blue                   ;; farve
    set size 1.5                     ;; størrelse
    setxy random-xcor random-ycor    ;; position
    set person-infection-time 0      ;; infektionstid
  ]
  ask n-of 3 persons [set color red]  ;; antal personer der initialt er smittet
end

to random-walk
  ask persons [
    right random 30
    left random 30
    forward 0.1
  ]
end

to transmit
  ask persons with [color = blue] [
    if any? persons-here with [color = red] [
      set color red
    ]
  ]
end

to increment-time-infected
  ask persons with [color = red] [
    set person-infection-time person-infection-time + 1
  ]
end

to become-immune
  ask persons with [color = red] [
    if person-infection-time > expected-infection-time [
      set color yellow
    ]
  ]
end
```

Figur 10: Fanebladet Code for smittemodellen.

Når man skal læse programkode, er det derfor ofte en god idé at veksle mellem en top-down og en bottom-up fremgangsmåde. Top-down forstået på den måde, at man forsøger at forstå, hvad programmet overordnet forsøger at opnå, og hvordan dette udføres. Bottom-up forstået på den måde, at man forsøger at forstå, hvad de enkelte delelementer i programmet er, hvad deres rolle er og hvordan de interagerer med andre elementer. Vi demonstrerer denne fremgangsmåde for kodeforståelse i de kommende afsnit.

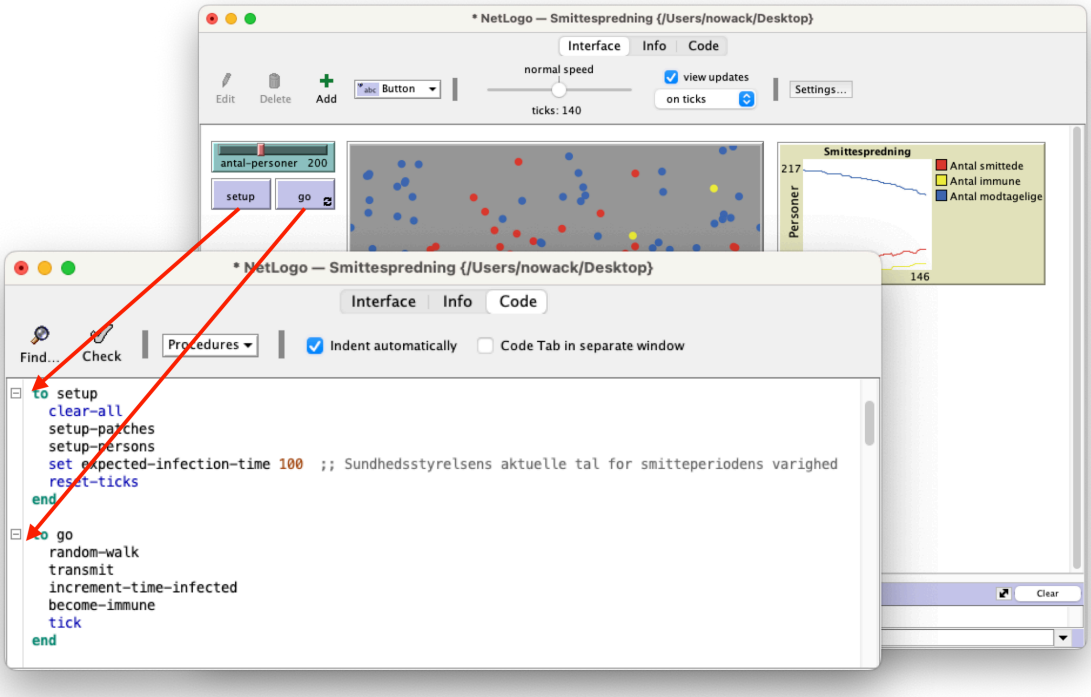
Hvis man kigger på koden i figur 10, observerer man hurtigt, at det meste af koden er fordelt på 8 klumper, der alle sammen begynder med nøgleordet "to", altså at gøre (et eller andet) og afsluttes med nøgleordet "end". Her er tale om 8 procedurer, der alle sammen kan kaldes af computeren og dermed udføre noget. Det, der udføres, er i hvert tilfælde koden i mellem "to"-nøgleordet og "end"-nøgleordet. En procedure er en måde at give en samling instruktioner et navn, som man så kan kalde fra et andet sted i programmet. Det giver to fordele:

- Koden bliver nemmere at læse og forstå, fordi den bliver delt op i mindre bidder, der hver især er nemmere at overskue end den samlede helhed.
- Hvis man oplever, at man skriver den samme kode igen og igen, så er det nemmere at putte koden ind i en procedure og så kalde den fra de ønskede steder. Det har også den fordel, at hvis man senere skal rette i koden eller udvide koden, så foregår det kun dette ene sted.

Et typisk designmønster for NetLogo-modeller er, at de indeholder en "setup"-procedure og en "go"-procedure. "Setup"-proceduren gør modellen klar, dvs. sletter evt. output fra forrige kørsel, nulstiller værdien af variable, tegner den verden, som agenterne skal bevæge sig rundt i, skaber og placerer de agenter, der skal være i modellen fra start, samt nulstiller modellens klokke. "Go"-proceduren får modellen til at køre og gentages (dvs. den kaldes igen og igen af computeren) indtil modellen er færdig eller manuelt stoppes.

2.1.3. Sammenhæng mellem grænsefladen og koden

Som tidligere nævnt, så er det gennem Interfacefanebladet, at vi interagerer med modellen. Sammenhængen med koden er helt simpel, se figur 11. De røde pile indikerer at, når man trykker på "setup"-knappen, bliver koden i proceduren "setup" udført, og når man trykker på "go"-knappen, bliver koden i proceduren "go" udført.



Figur 11: Knapperne i interfacet kalder procedurer i koden.

Bemærk dobbeltpilene på "go"-knappen. Som det fremgår har "setup"-knappen ikke et tilsvarende symbol på sig. Dobbeltpilene indikerer, at "go"-knappen er lavet som en "forever"-knap, dvs. at koden for denne knap bliver kaldt igen og igen af computeren. Med andre ord, når "go"-proceduren udføres af computeren, og den når til sidste linje ("end"), så hopper computeren op til starten af proceduren og udfører det samme en gang til. Koden i "go"-proceduren bliver således udført "for evigt". Der er dog tre måder at stoppe denne uendelige løkke på. Det mest normale er, at man blot trykker på "go"-knappen igen, hvorefter koden og modellen sættes på pause. Man kan også indsætte en linje kode i "go" proceduren, der stopper udførslen når en eller anden betingelse er opfyldt (f.eks. efter 5000 ticks eller når der er 350 immune personer). Endelig kan man altid stoppe en simulering ved at benytte NetLogos menu:

Øvelse 5: At stoppe en simulering vha. menuen.

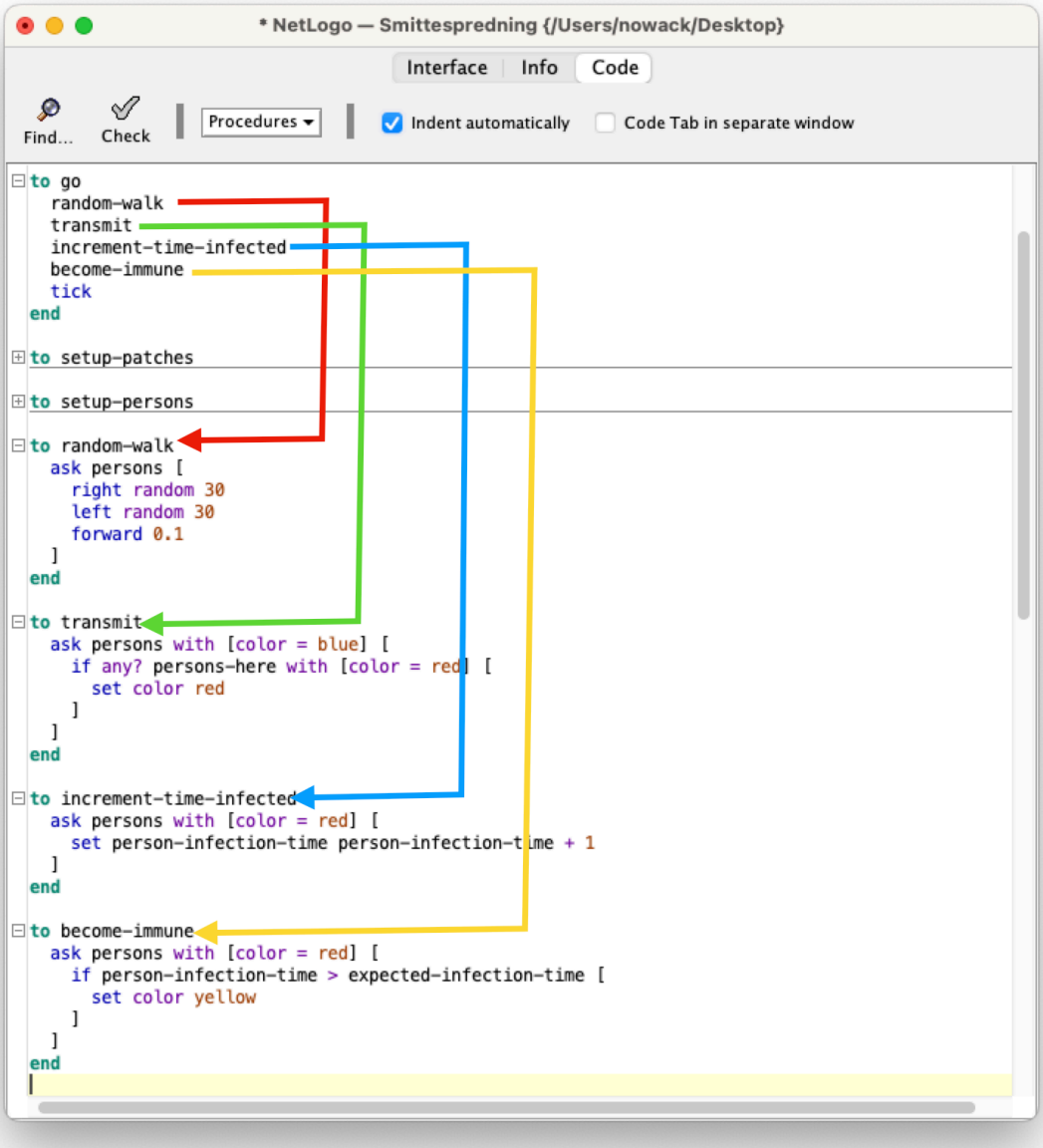
- Åben smittespredningsmodellen i NetLogo.
- Sæt modellen til at køre.
- Vælg punktet "Halt" i NetLogo's "Tools" menu.

2.1.4. Flere procedurer i koden

Vi er allerede stødt på to procedurer i foregående afsnit, nemlig "setup" og "go" procedurerne, der kaldes via knapperne i grænsefladen. Alle andre procedurer kommer først i spil, når de bliver kaldt inde fra selve koden.

Figur 12: "setup" proceduren kalder to andre procedurer.

I figur 12 er det illustreret, hvordan "setup"-proceduren kalder to andre procedurer: nemlig "setup-patches" og "setup-persons". Det skal forståes på den måde, at når computeren kommer til det sted i koden, hvor der står "setup-patches" (dvs. i "setup"-procedurens 3. linje), så hopper den ned til proceduren, der begynder med "to setup-patches" og udfører disse linjer kode en efter en, indtil den møder nøgleordet "end", hvorpå den hopper tilbage til "setup" proceduren og fortsætter med næste linje. I dette til fælde er næste linje "setup-persons", og dermed hopper computeren ned til definitionen af denne procedure (dvs. "to setup-persons") og udfører disse linjer kode. Når computeren igen rammer nøgleordet "end" afsluttes proceduren, og den vender tilbage til "setup"-proceduren og fortsætter med næste linje, hvilket i dette tilfælde er linjen "set expected-infection-time 100", hvis betydning vi vender tilbage til.



Figur 13: “go”-proceduren kalder fire andre procedurer.

I figur 13 er det tilsvarende vist, hvordan “go”-proceduren kalder de fire procedurer, der tilsammen udgør adfærdsreglerne for alle agenterne. Bemærk også de små firkanter med “+” og “-“ til venstre for hver procedure. Hvis man klikker på en firkant med “-” kollapses proceduren, så man kun kan se procedurens navn (og symbolet ændres til en firkant med et “+”). Hvis man omvendt trykker på en kollapset procedures “+” symbol, så sker det modsatte: koden for proceduren vises, og symbolet ændres til en firkant med et “-“ igen. Det er vigtigt at forstå, at disse handlinger ikke sletter koden, men blot skjuler den. Det bruges til at gøre vinduet mere overskueligt, når man har mange procedurer med meget kode i.

Øvelse 6: At udvide og kollapse visningen af koden.

- Åben smittespredningsmodellen i NetLogo.
- Gå om på kodefanebladet.

- Prøv at kollapse og udvide dele af koden.

Sammenfattende har vi i dette afsnit demonstreret brugen af et særdeles anvendt princip i programmering, ofte kaldet "Divide and Conquer" eller "Main and Subroutines". Når man skal løse et komplekst problem, som f.eks. at simulere smittespredning i dette tilfælde, så kan man med fordel dele problemet op i mindre problemer og så løse de mindre problemer hver for sig, med anvendelsen af små delløsninger. I dette eksempel splittede vi adfærden for smittespredning op i 4 stumper deladfærd: bevæge sig tilfældigt rundt, bliv smittet, husk på hvor længe man har været smittet og bliv immun. Til hver af disse 4 delproblemer er der så lavet 4 delløsninger i form af 4 mindre stumper kode fordelt på 4 procedurer. Disse 4 procedurer (eller subroutiner) kaldes så fra en overordnet hovedprocedure (eller main), der også orkestrerer rækkefølgen af hvornår de 4 procedurer kaldes.

2.1.5. At ændre i koden

I det følgende skal du prøve at lave et antal ændringer i koden for smittemodellen.

Øvelse 7: Ændre agenternes udseende.
- Åben smittespredningsmodellen i NetLogo.
- Gå om på kodefanebladet.
- Find det sted i koden, hvor personerne får tildelt deres form ("set shape "dot").
- Prøv at ændre formen til en person ("set shape "person")
- Kør modellen igen og se ændringerne.

Øvelse 8: Ændre agenternes adfærd.
- Åben smittespredningsmodellen i NetLogo.
- Gå om på kodefanebladet.
- Find det sted i koden, hvor der testes på om personerne har været smittet i mere end 100 dage.
- Ændr dette til 50 dage.
- Kør modellen igen og se ændringerne.

Øvelse 9: Ændr hvor mange smittede, der er, når modellen startes.
- Åben smittespredningsmodellen i NetLogo.
- Gå om på kodefanebladet.
- Find det sted i koden, hvor der skabt tre smittede personer.
- Ændr dette til 10 personer.
- Kør modellen igen og se ændringerne.

Øvelse 10: Ændre agenternes farve.
- Åben smittespredningsmodellen i NetLogo.
- Gå om på kodefanebladet.
- Find det sted i koden, hvor personerne får tildelt deres farve ("set color blue").
- Prøv at ændre farven på en person til hvid ("set color white")
- Kør modellen igen og observer ændringen.
- Observer også den anderledes adfærd af smittemodellen. Tilsyneladende bliver ingen længere smittet. Hvorfor?

- Find det sted i koden, hvor personer bliver smittede. Ret koden, så den nu virker for hvide personer.

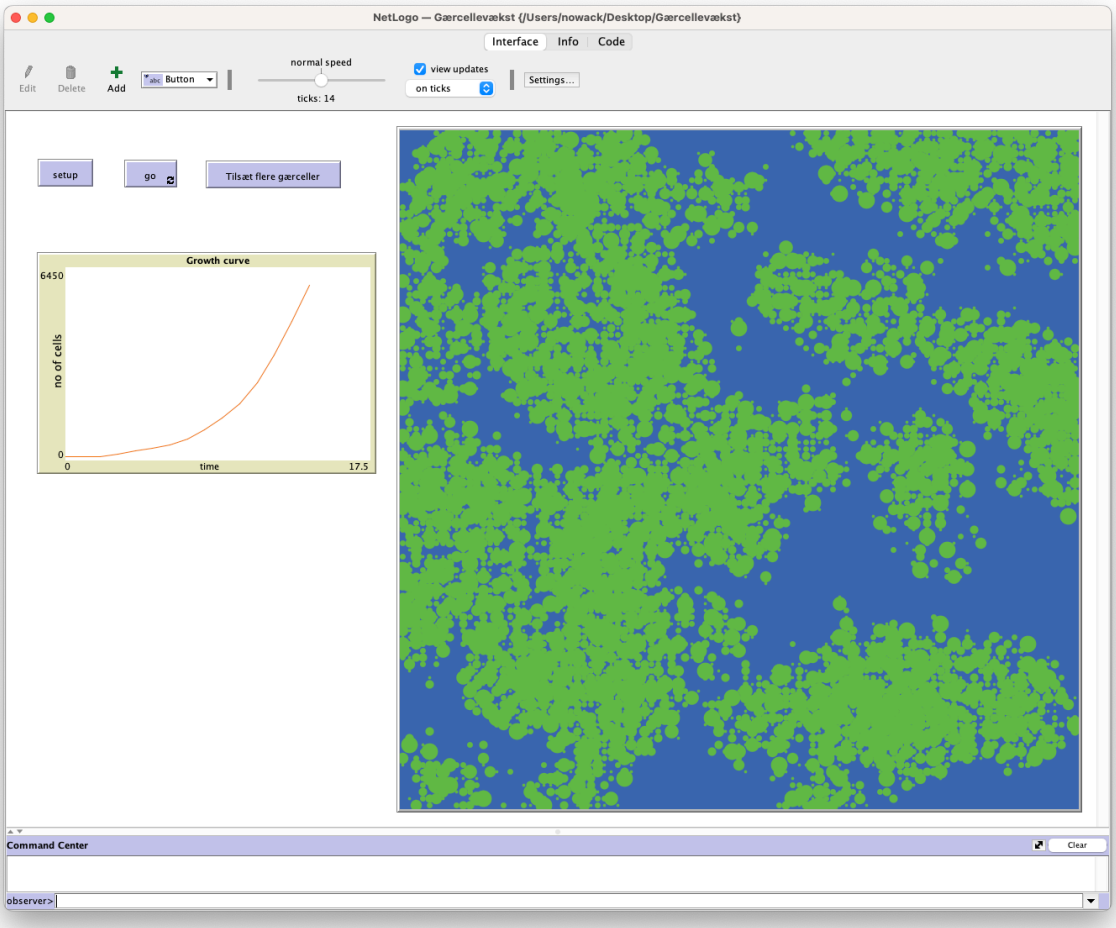
2.2. Gærcellemodel

I det følgende gennemgåes endnu et konkret eksempel på en model. Modellen er af gærceller og deres vækst. Formålet med eksemplet er at give dig endnu en rundtur i NetLogo-værktøjet, hvor vi dels repeterer enkelte emner, dels introducerer nye emner. Først gennemgåes den grænseflade hvormed man kan interagere med modellen. Dernæst beskrives det faneblad, hvor man beskriver modellen. Herefter kaster vi os igen over koden: vi beskriver sammenhængen mellem grænsefladen og koden, de grundliggende agenter i NetLogo, der kaldes "turtles" og "patches", og endeligt de 4 procedurer, der implementerer adfærdsreglerne for denne models agenter.

Øvelse 11: Orienter dig i gærcellemodellens interface.
- Åben "Gærcellevækst.nlogo" i NetLogo.
- Eksperimenter med "setup" og "go" knapperne.
- Eksperimenter med knappen "Tilsæt flere gærceller"

2.2.1. Interface-Tab

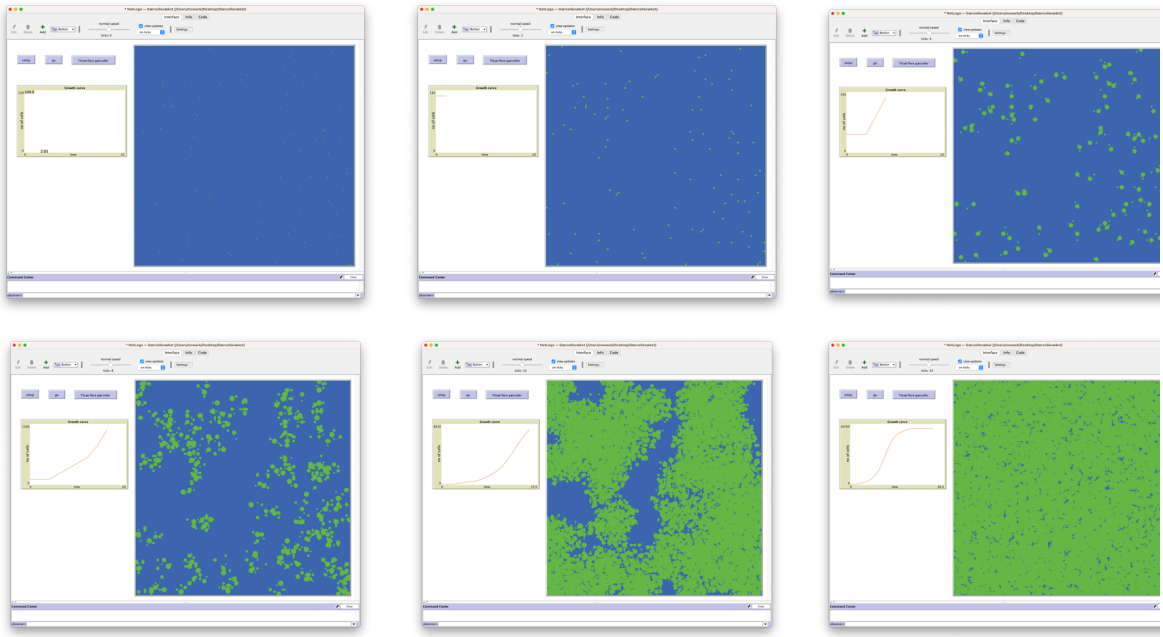
Figuren nedenfor viser NetLogo-værktøjet, efter at man har loadet modellen, og modellen har kørt i et ganske lille tidsrum. Til højre er en grafisk visning af modellens aktuelle tilstand. De grønne pletter er gærceller. Til venstre er der placeret tre knapper og en graf (kaldet et plot i NetLogo). Knapperne hedder "setup", "go" og "Tilsæt flere gærceller".



Figur 14: NetLogo værktøjet med en indlæst model.

Igen trykker vi på "setup"-knappen for at gøre modellen klar til udførelse. Herefter trykker vi på "go"-knappen, for at få modellen til at køre. Når modellen kører, vil den grafiske visning ændre sig, og man ser udviklingen blandt gærcellerne. Hvis man trykker på "go"-knappen igen, stoppes udførslen. Grafen viser udviklingen i antallet af gærceller over tid. Den ekstra knap "Tilsæt flere gærceller" kan man trykke på, hvis man manuelt ønsker at tilsætte flere gærceller undervejs, og de vil så dukke op i grafikken som nye grønne pletter.

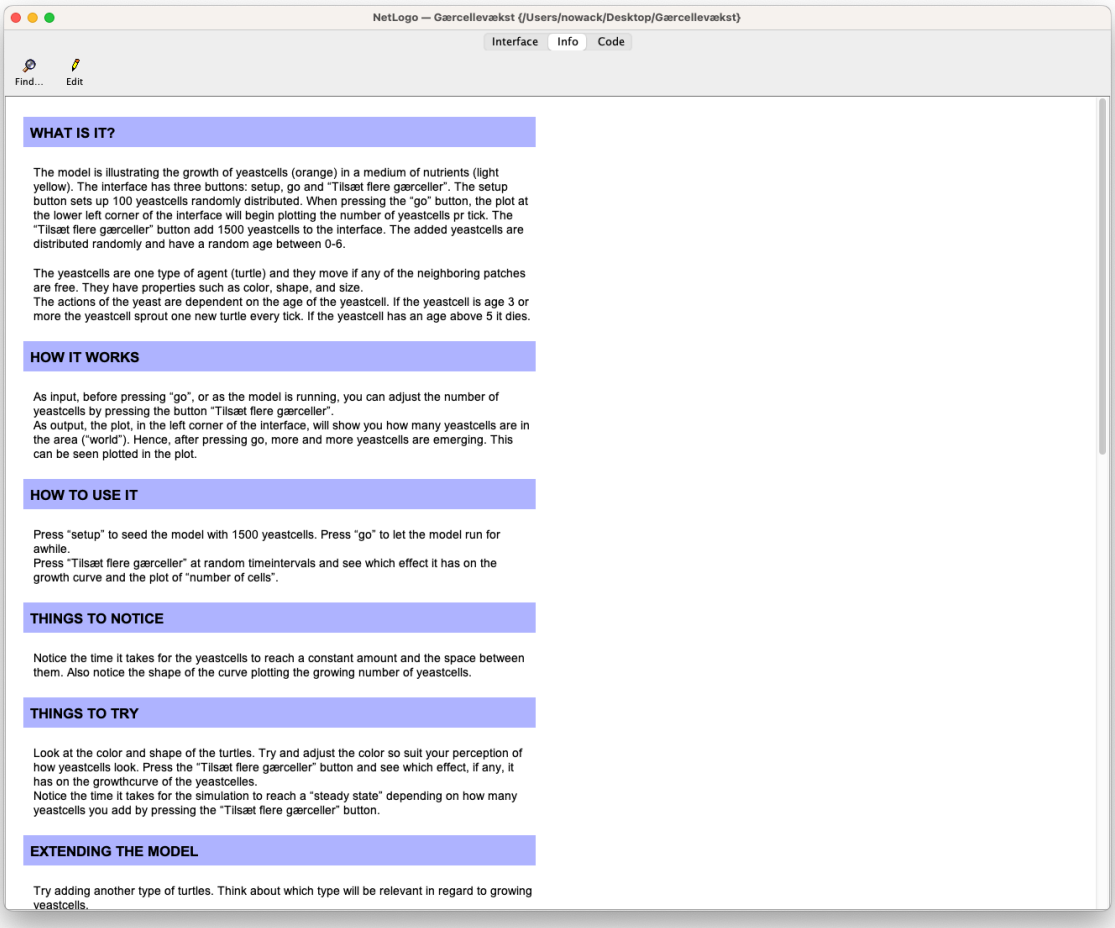
Nedenfor illustreres hvordan modellen kan se ud efter forskellige tidsperioder.



Figur 15: Gærcellemodel I NetLogo efter forskellige antal tidsskridt.

2.2.2. Info-Tab

Da vi gennemgik Smittemodellen, beskrev vi indholdet af Interface og Code fanebladene, men vi sprang over det midterste faneblad, "Info". Hvis man klikker på Info-knappen, kommer man over på Info-fanebladet. Her har udvikleren beskrevet modellen. Der står typisk, hvilke fænomener modellen omhandler, hvordan den virker, hvordan man bruger den, ting man skal være opmærksom på, ting man kan prøve, forslag til udvidelser og ændringer af modellen, samt evt. referencer til relevante tekster, andre modeller, online videoer osv.



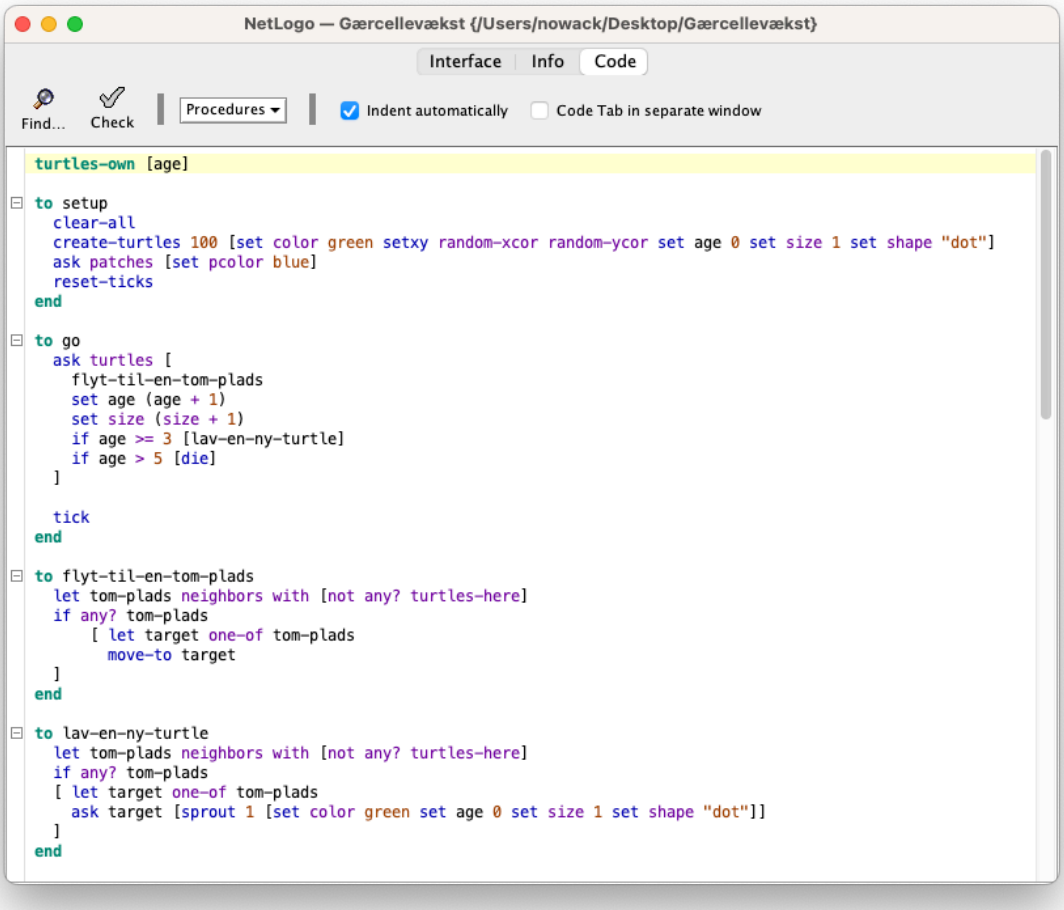
Figur 16: Info fanebladet for Gærcellevækstmodellen i NetLogo.

Øvelse 12: Info fanebladet

- Åben gærcellemodellen i NetLogo.
- Orienter dig i Info fanebladet.

2.2.3. Code-Tab

Lad os hoppe om på fanebladet med koden for denne model.



Figur 17: Code fanebladet i NetLogo værktøjet.

I ovenstående program ser man hurtigt, at koden primært er fordelt på 4 procedurer. Man kan også observere, at nøgleordet "if" benyttes nogle gange. IF-konstruktionen betyder, at HVIS en betingelse er opfyldt (f.eks. noget med "age", alder), så vil en bestemt linje kode blive udført. Ellers ikke.

Desuden ser man også at nøgleordet "set" bliver brugt. F.eks sættes nogle farver "color" og "pcolor" til værdierne grøn og blå henholdsvis. Det svarer til farverne på den grafiske visning på Interface fanebladet.

Øvelse 13: At ændre farven på agenter vha koden.

- Åben gærcellemodellen i NetLogo.
- Gå om på kodefanebladet.
- Find linjen, der indeholder koden "set color green"
- Ændr koden til at være "set color red"
- Gå tilbage til Interface fanebladet og tryk på "setup" knappen. Observer forskellen.
- Eksperimenter med forskellige farver på agenterne.

Øvelse 14: At ændre farven på baggrunden vha koden.

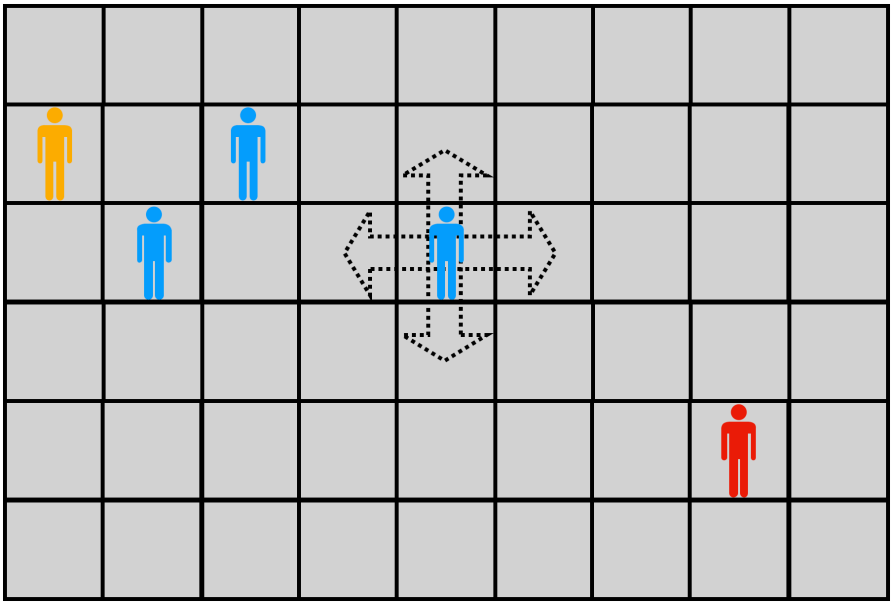
- Åben gærcellemodellen i NetLogo.
- Gå om på kodefanebladet.

- Find linjen, der indeholder koden “set pcolor blue”
- Ændr koden til at være “set pcolor white”
- Gå tilbage til Interface fanebladet og tryk på “setup” knappen. Observer forskellen.
- Eksperimenter med forskellige farver på baggrunden. Prøv f.eks. at sætte farven på agenterne og baggrunden til den samme farve.

Endelig vil man nok bemærke, at nøgleordene “turtle” og “patch” går igen flere steder. Det kræver en nærmere forklaring, som følger i næste afsnit.

2.2.4. Intermezzo: turtles & patches

Som nævnt er NetLogo et agentbaseret modelleringsværktøj, dvs. at den primære abstraktion er en agent. En agent er en entitet, der har tilstand og adfærd (dvs. kan følge/udføre instruktioner). Turtles og patches er to forskellige typer af agenter i NetLogo. Turtles er agenter, der kan bevæge sig rundt i det, der i NetLogo kaldes for en “verden”. Patches er de felter, der udgør denne verden, se figur 18. I figuren er de grå felter patches, og de små mennesker er turtles.



Figur 18: “Patches” og “turtles” i en NetLogo “world”.

Patches kan ikke flytte sig, men de kan have både tilstand og adfærd. F.eks. har de en farve og en størrelse, og de kan f.eks. have den adfærd, at alle turtles, der kommer på besøg på patchen, bliver bedt om at skifte farve til lyserød, størrelse til det dobbelte eller flytte sig væk fra patchen igen.

Turtles kan bevæge sig (i figuren vist med de 4 pile: op, ned, til højre og til venstre, men de kan flytte sig i alle retninger og alle distancer — f.eks. 17 pixels i retningen 43 grader).

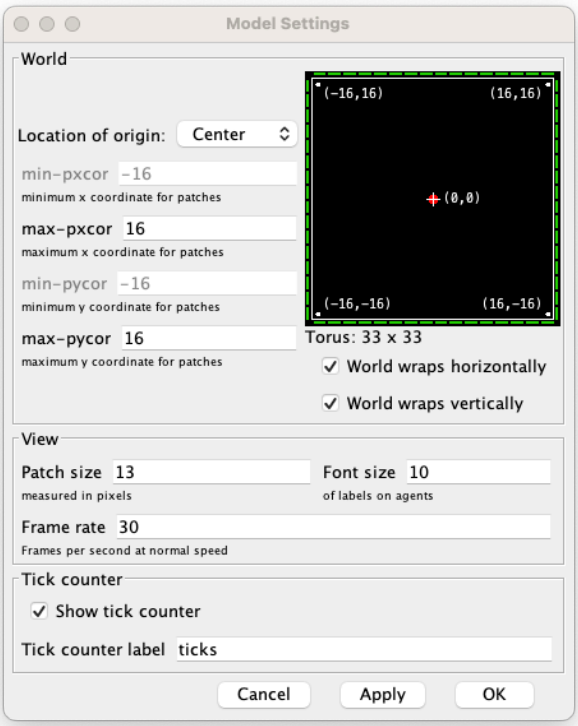
En NetLogo-verden har dimensioner bestående af antal patches i den lodrette akse samt antal patches i den vandrette akse. F.eks. består verden i figuren af en verden på 9x6 patches. Dette giver et koordinat system, som man kan bruge i sin programkode. Man kan selv bestemme, hvordan koordinatsystemet skal organiseres. F.eks. kan man vælge at lokationen

(0,0) skal være feltet, hvor den midterste blå person er placeret. I dette tilfælde vil den røde person være placeret på lokationen (3, -2). Man kan også vælge, at lokationen (0,0) skal være øverste venstre hjørne. I dette tilfælde vil den røde person være placeret på position (7, -4). Eller man kan vælge at lokationen (0,0) skal være nederste højre hjørne. I dette tilfælde vil den røde person være placeret på position (-1, 1).

Man bestemmer ligeledes selv størrelsen på både patches og turtles, og man bestemmer selv størrelsen på verden, om det skal være 9x6 patches som i figuren eller f.eks. 500x200 patches.

Endelig bestemmer man også selv, om modellen skal være endelig eller med "wrap-around". I det første tilfælde kan den røde person i figuren kun tage et enkelt helt skridt til højre, hvorefter vedkommende kun kan gå til venstre (tilbage igen), op eller ned, men altså ikke "ud over kanten". Hvis man derimod indstiller sin verden til at have "wrap around" vil den røde person, hvis vedkommende går to skridt til højre, dukke op på første felt i modsatte side.

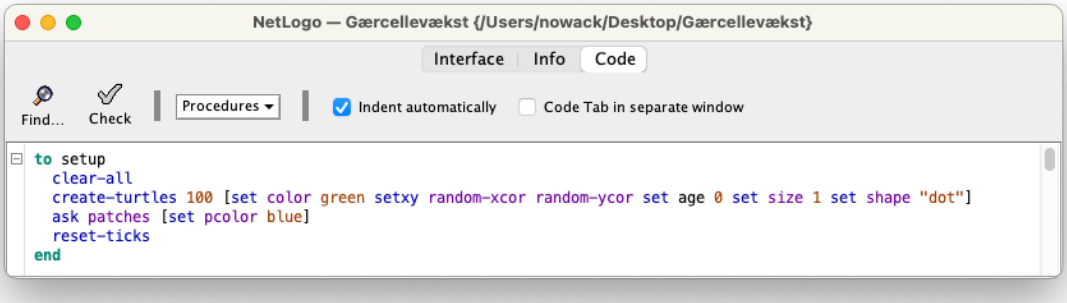
Disse indstillinger til en models verden foretages i Interface fanebladet, ved at trykke på knappen "Settings…", hvorpå vinduet i figur 19 fremkommer.



Figur 19: Indstillinger for verden i NetLogo.

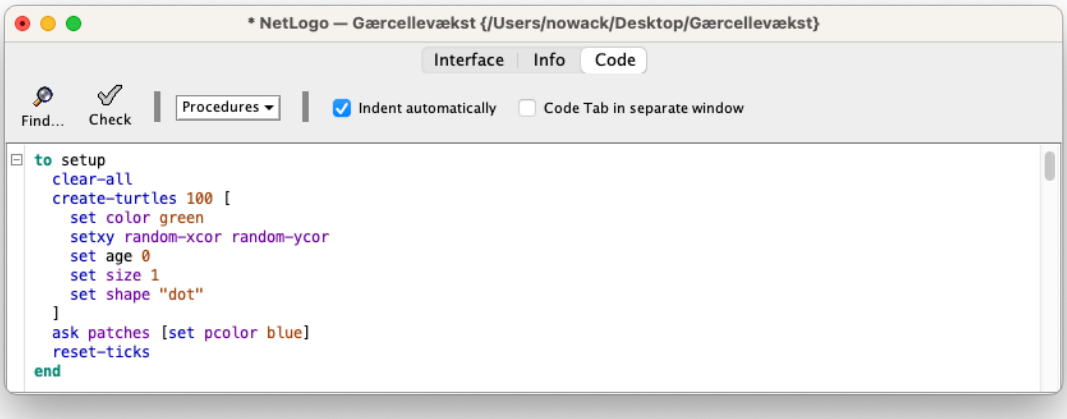
2.2.5. Setup-proceduren

Efter dette lille intermezzo om turtles og patches er vi klar til at kigge lidt nærmere på mere af koden i eksemplet. Vi starter med "setup" proceduren, se figur 20



Figur 20: Setup proceduren for gærcellemodellen.

"Setup" proceduren nævner både turtles og patches. Lad os starte med at omformatere koden, så den bliver lidt nemmere at læse (vi indsætter bare nogle linjeskift - det påvirker ikke, hvad koden betyder og gør, når den bliver kaldt), se figur 21.



Figur 21: Setup proceduren for gærcellemodellen omformateret.

Setup proceduren består af 4 klumper kode i mellem to og end nøgleordene.

Den første linje "clear-all" sletter resterne fra eventuelle foregående udførelser af modellen, herunder at alle "gamle" patches og turtles slettes.

Den næste klump kode skaber 100 turtles, og derpå udfører den al den kode, der står i mellem de kantede parenteser [] for hver eneste turtle. Dvs. at for den enkelte turtle:

- bliver farven sat til grøn
- x og y koordinaterne (dvs. placeringen i verden) bliver sat til tilfældige værdier
- variable "age" bliver sat til 0
- størrelsen af turtlen bliver sat til 1
- formen af turtlen bliver sat til noget, der kaldes en "dot", hvilket blot er en farvet cirkel.

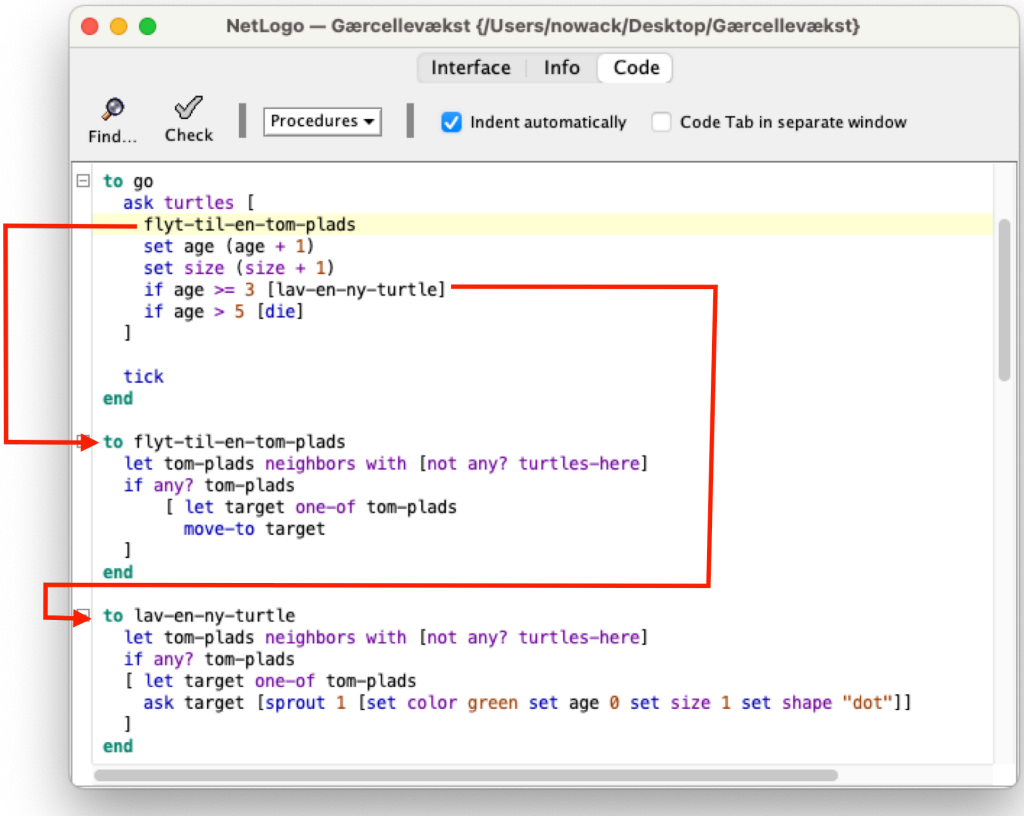
Bemærk, at fordi at al koden mellem de kantede parenteser bliver udført for hver enkelt turtle, så vil x- og y-koordinaterne for hver enkelt turtle være tilfældige og sandsynligvis forskellige. Dvs. at placeringen af hver enkelt turtle er tilfældig.

Den tredje klump kode ("ask patches …") beder alle patches om at sætte deres farve til blå. Dette gøres ved at sætte værdien af variablen "pcolor" til "blue". Alle patches har en indbygget variabel, der hedder "pcolor", ligesom de alle har en størrelse, et x- og et y-koordinat.

Endelig udføres kommandoen "reset-ticks", der sætter systemets indbyggede klokke eller tæller, om du vil, til 0. Dette kræver en nærmere forklaring. NetLogo-modeller anvender en tidsenhed, der kaldes et "tick" (i stil med sekunder og minutter). Når man skriver kommandoen "tick" i sin kode, så tælles "tick" tælleren op med 1 i systemet (se det efterfølgende afsnit om "go" proceduren). Dvs. hvis "tick" tælleren før havde værdien 3, så vil den efter kommandoen "tick" have værdien 4. Kommandoen "reset-ticks" nulstiller "tick" tælleren til … ja, "0". Man kalder normal "reset-ticks" kommandoen som det sidste, man udfører i sin "setup" procedure. Dvs. at man gør sin model klar med forskellige kommandoer, som dem vi netop har gennemgået, og til sidst sætter man så klokken til at være "0".

2.2.6. Go-proceduren

Den næste procedure i koden er "go-proceduren", se figur 22. Den består overordnet af to dele: en første - lidt større - del, der beder alle turtles om at udføre en samling kommandoer; en anden, der blot tæller "tick" tælleren op med 1. Bemærk de to røde pile: de angiver, at man inde fra "go"-proceduren på bestemte tidspunkter kalder to andre procedurer; nemlig proceduren "flyt-til-en-tom-plads" og "lav-en-ny-turtle".



Figur 22: Go proceduren for gærcellemodellen.

Som nævnt er det første, der sker, når "go" proceduren bliver kaldt, at vi beder alle turtles om at udføre en række kommandoer. De er som følger:

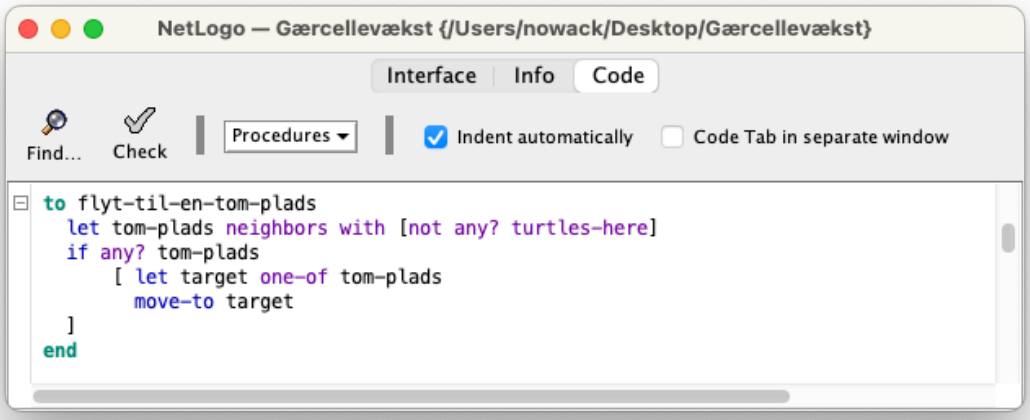
- turtlen kalder proceduren "flyt-til-en-tom-plads". Den vender vi tilbage til, men groft sagt, så beder vi den enkelte turtle om at flytte hen på en patch, hvor der ikke i forvejen står andre turtles.
- turtlen ændrer sin egen alder, dvs. den sætter alderen, til det den var forvejen, plus 1.
- turtlen ændrer sin egen størrelse, dvs. den sætter størrelsen, til det den var i forvejen, plus 1.
- nu skal turtlen foretage to valg. Det angiver man i NetLogo med "if" nøgleordet. En "if" sætning består af nøgleordet "if" efterfulgt af en betingelse og en kommandoblok. En betingelse er noget, der kan evalueres til enten sandt eller falsk. Hvis betingelsen er sand, så udføres kommandoblokken. Hvis betingelsen er falsk, så udføres kommandoblokken ikke. En kommandoblok er alt det, der står mellem de kantede parenteser. Dvs. at de næste to linjer betyder:
  - Hvis værdien af variablen "age" er større end eller lig med 3, så kaldes proceduren "lav-en-ny-turtle". Den vender vi tilbage til, men sammenfattende så skaber modellen en ny turtle på en af de patches, der er ved siden af den patch, hvorpå den turtle vi taler med lige nu står.
  - Hvis værdien af variablen "age" er større end 5, så kaldes kommandoen "die" på den turtle vi taler med i øjeblikket. Når en turtle udfører

kommandoen “die”, så slettes den fra modellen, dvs. at den forsvinder.

Når hver eneste turtle har udført disse 5 kommandoer, så beder vi systemet opdatere klokken for modellen ved at kalde kommandoen “tick”.

2.2.7. Proceduren: flyt-til-en-tom-plads

Vi mangler at forklare to procedurer i programmet. Modsat “setup” og “go”, så kaldes de ikke fra knapper i interfacet, men derimod inde fra selve “go” proceduren. Koden for den første er gengivet i figur 23.



Figur 23: “flyt-til-en-tom-plads” proceduren for gærcellemodellen.

Det første, der sker er, at vi med kommandoen “let” erklærer en lokal variabel “tom-plads” og giver den en værdi. At variablen er lokal betyder, at variablen kun kan bruges indefra denne procedure. Når proceduren er udført, så er variablen ikke længere tilgængelig og kan derfor ikke benyttes længere. Dette er modsætning til de variable, der er erklæret på agentniveau, som f.eks. “pcolor” for patches og “size” og ”age” for turtles. Sådanne agent-variable kan altid tilgåes (benyttes) fra den enkelte agent og de bevarer deres værdi, selv når vi ikke lige bruger dem.

Tilbage til “let” kommandoen. “Let” tager to parametre. Den første er navnet på variablen - i dette tilfælde har vi valgt at kalde den for “tom-plads” . Den anden er den værdi, vi ønsker at tildele variablen. I dette tilfælde er det udtrykket “neighbors with [not any? turtles-here]”. Det kræver en nærmere forklaring, og det er nemmest at forstå, hvis vi læser udtrykket bagfra, dvs. fra højre mod venstre:

- “turtles-here” er en kommando i NetLogo, der returnerer en mængde bestående af alle de turtles, der står på samme plads, som den der kalder kommandoen (inklusive turtlen selv, hvis det er en turtle, der kalder kommandoen).
- “any?” er en kommando, der tager en enkelt parameter, nemlig en mængde af agenter. Hvis mængden ikke er tom (dvs. hvis der er nogle agenter i mængden), så returnerer kommandoen værdien “sandt”. Hvis mængden er tom, så returnerer den værdien “falsk”.

- "not" er en simpel kommando, der tager en enkelt parameter, som skal være et udtryk, der enten er sandt eller falsk. Hvis udtrykket er "sandt", så returnerer den "falsk", og hvis udtrykket er "falsk", så returnerer den "sandt". Med andre ord, så returnerer den præcist det modsatte af det den bliver kaldt på.
- Sammenfattende så kan udtrykket i de kantede parenteser oversættes til: "alle de agenter, hvorpå der ikke står en turtle."
- "with" er en kommando, der også tager to parametre, men modsat de andre kommandoer, vi har kigget på indtil videre, så står den første parameter til venstre for "with" og den anden parameter står til højre for "with". Den første parameter skal være være en mængde af agenter (normalt turtles eller patches), og den anden parameter skal være en betingelse, der kan evaluere til sandt eller falsk for hver enkelt agent. Det som "with" returnerer er så mængden af alle de agenter fra agentmængden i første parameter, hvor betingelsen i anden parameter evaluerer til "sandt". I dette konkrete tilfælde hvor den første parameter er "neighbors" og den anden parameter er "[not any? turtles-here]", så returnerer kommandoen alle de nabopatches, hvor der ikke står en eller flere turtles i forvejen.
- Dvs. sammenfattende så tildeles variablen "tom-plads" mængden af alle de nabopatches, hvor der ikke i forvejen står en eller flere turtles.

Ovenstående kan ved første øjesyn forekomme utroligt kompliceret. På den anden side, så lærer man ret hurtigt en mængde af disse kommando-kombinationer, når man går i gang med at forstå, ændre og udvide eksisterende modeller. Det bliver en del af ens ordforråd som "modulør" på samme niveau som de enkelte kommandoer i NetLogo. Det illustrerer også to store fordele ved NetLogo programmeringssproget: dels, at man med meget få kommandoer kan udtrykke noget temmelig kompliceret; dels at selvom det kan være svært de første gange at stykke sådan en kommando sammen på egen hånd, så er det faktisk ikke så svært at læse, da betydningen af konstruktionen ligger meget tæt op ad naturligt (ok, lidt kluntet) sprog: "naboer med ikke nogen turtles på".

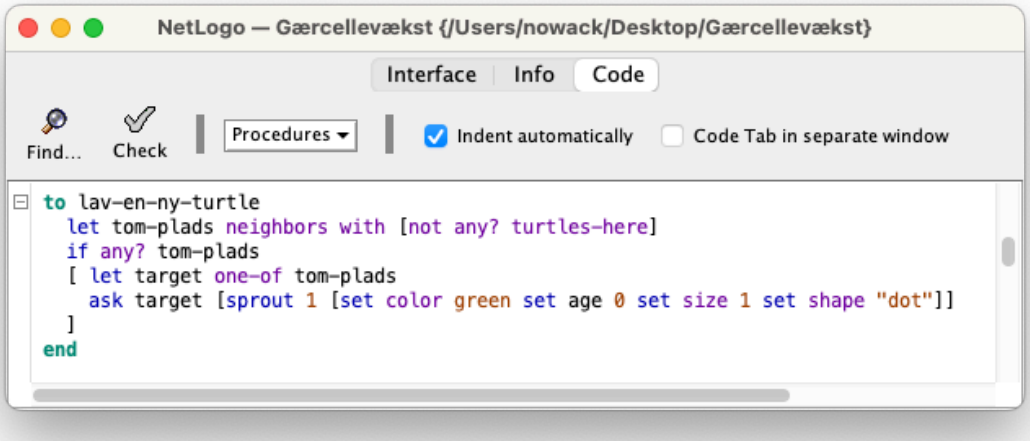
Det næste der sker er, at vi udfører en if-sætning. Den kender vi. Efter "if" kommer en betingelse, der skal være opfyldt, hvis kommandoblokken mellem de kantede parenteser skal udføres. I dette tilfælde er betingelsen "any? tom-plads", og vi husker fra før, at "any?" returnerer "sandt" hvis mængden ikke er tom. Dvs. hvis der ER nogle agenter i variablen "tom-plads", så udfører vi kodeblokken. Kodeblokken gør to ting:

- Vi erklærer en ny lokal variabel, kaldet "target" og tildeler den værdien "one-of tom-plads." Vi husker, at "tom-plads" er mængden af nabopatches, hvorpå der ikke står en turtle i forvejen. Kommandoen "one-of" giver os ganske simpelt én af disse, tilfældigt udvalgt.
- Kommandoen "move-to" tager en enkelt parameter (en agent), og gør ikke overraskende det, at den flytter vores turtle (den som udfører denne kode), til den agent, der står efter kommandoen. I dette tilfældet er parameteren "target" fra ovenstående linje, dvs. en tilfældig tom nabopatch.

Voila! Sammenfattende: Alle turtles, der kalder denne kode, flytter sig selv hen på en tom naboplads, hvis en sådan findes.

2.2.8. Proceduren: lav-en-ny-turtle

Vores models sidste procedure er næsten magen til den foregående. Det er proceduren "lav-en-ny-turtle", der er vist i figur 24. Den følger samme melodi, dvs. at den laver en variabel, der gemmer mængden af tomme nabopladser (hvis der er nogen). Herefter følger en if-sætning, der udføres, hvis der er nogle tomme nabopladser. Hvis det er tilfældet, så udvælges igen en tilfældig af de tomme nabopladser.



Figur 24: "lav-en-ny-turtle" proceduren for gærcellemodellen.

Men herefter adskiller koden sig fra den foregående procedure. I stedet for at flytte sig selv til den valgte plads som før, så beder vores turtle nu den tilfældigt udvalgte naboplads om hjælp:

- Kommandoen "sprout" bruges til at skabe ny turtles (og ikke nødvendigvis rosenkål). Kommandoen tager to parametre: den første er antallet af nye turtles, der skal skabes; den anden er en kodeblok, der udføres for alle disse nye turtles.
- Dvs. at i dette tilfælde kaldes "sprout" på den udvalgte naboplads og der skabes én ny turtle på denne, hvorefter kommandoerne mellem de kantede parenteser udføres af denne nye turtle. Kommandoerne er:
    - sæt farven til grøn
    - sæt alderen til 0
    - sæt størrelsen til 1
    - sæt formen til en "dot"

Voila! Alle turtles, der udfører denne kode, vil skabe en ny turtle på et ledigt nabofelt, hvis et sådan eksisterer.

2.2.9. Den decentraliserede tankegang – del 2

En vigtig indsigt man kan få, når man læser koden for gærcellemodelen, er at alle turtles (dvs. gærceller) har en alder. Den starter med at være 0, når turtlen bliver skabt, men for hver gang "go" proceduren kaldes, så bliver alderen 1 større. Når alderen er 3, 4, eller 5, så

skaber gærcellen en ny gærcelle ved siden af sig selv (hvis der er plads). Denne nye gærcelle har så alderen 0 til at begynde med. Det betyder også, at hver enkelt gærcelle i princippet kan skabe 3 andre nye gærceller, inden den fylder 5 og dør.

Bemærk også hvordan vi på simpel vis kan modellere, at en gærcelle vokser ved blot at øge dens størrelse for hver gang "go" proceduren kaldes. Det betyder i praksis, at en gærcelle starter med at have størrelsen 1, og så løber den igennem størrelserne 2, 3, 4 og 5 inden den dør og forsvinder.

Sammenfattende demonstrerer ovenstående det kraftfulde ved den agentbaserede modellerings princip om lokalitet (den decentraliserede tankegang). Ved at skrive koden med udgangspunkt i den enkelte agent, kan koden holdes relativt simpel. Alle agenter udfører præcis den samme kode, men fordi de har forskellig tilstand (i dette tilfælde alder, størrelse og placering), så ender de med at have have forskellige aldre og størrelser, når modellen har været kørt blot ganske få gange pga. if-sætningen, der afgør om de kan skabe nye gærceller ved siden af sig selv.

3. Ressourcer

Denne note har forhåbentligt givet inspiration og viden nok til at læseren har fået lyst til at arbejde videre med agentbaseret modellering i NetLogo. I dette afsnit vil vi derfor kort nævne nogle muligheder for at opsøge mere viden om begge dele.

Den primære reference til NetLogo systemet er CCL's egen webside: NetLogo. Vi anbefaler, at man orienterer sig i det nyere læringsmateriale, der hedder The Beginner's Guide to NetLogo Programming, der indeholder videointroduktion til flere modeller og mange af NetLogo-programmeringssprogets primitiver. Dette kan suppleres med de 3 tutorials, der er indeholdt i NetLogo User Manual. Endelig anbefaler vi, at man orienterer sig i NetLogos indbyggede modelbibliotek, hvilket introduceres i denne video: Exploring the Models Library.

I den mere langhårede afdeling findes der en mængde forskningsartikler og lærebøger om agentbaseret modellering og NetLogo. Hvis man er interesseret i de generelle sammenhænge mellem videnskabelig metode, brugen af modeller og læring, så kan det anbefales at kigge på (Box, 1976), og den decentraliserede tankegang, der er så fundamental for agentbaseret modellering, uddybes i (Resnick, 1996).

Hvis man er interesseret i fordelene ved anvendelsen af computerbaserede modeller kan man kaste et blik på (Nowack & Caspersen, 2014). Dette uddybes i relation til STEM-fagene i gymnasiet i denne korte video: Computational Thinking og Modellering i STEM-fag i gymnasiet. Endvidere uddybes rollen af agentbaseret modellering i relation til uddannelse og undervisning i denne video: Agent-based modelling in education og i (Weintrop et al., 2016).

Den definitive reference til agentbaseret modellering med NetLogo er (Wilensky & Rand, 2015). Hvis man er interesseret i en humaniora-vinkel på det samme, så er (Romanowska et al., 2021) en glimrende indgangsvinkel.

Referencer

Box, G. E. P. (1976). Science and Statistics. *Journal of the American Statistical Association*, *71*(356), 791–799. https://doi.org/10.2307/2286841
Nowack, P., & Caspersen, M. E. (2014). Model-based thinking and practice: A top-down approach to computational thinking. *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, 147–151. https://doi.org/10.1145/2674683.2674686
Resnick, M. (1996). Beyond the Centralized Mindset. *The Journal of the Learning Sciences*, *5*(1), 1–22.
Romanowska, I., Wren, C. D., & Crabtree, S. A. (2021). *Agent-based modeling for archaeology: Simulating the complexity of societies*. SFI Press.
Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology*, *25*. https://doi.org/10.1007/s10956-015-9581-5
Wilensky, U., & Rand, W. (2015). *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*. The MIT Press. https://www.jstor.org/stable/j.ctt17kk851