Galapagos

K-opgave, OOPD

Christian Stefansen

Fredag den 12. januar, 2007

Resumé

Dav, og velkommen i det grønne! I er netop blevet ansat som programmører i en verdensberømt forskningsgruppe af biologer, der arbejder på at belyse og forstå adfærdsmønstre fra et spilteoretisk synspunkt. Jeres første opgave er at implementere en model af adfærdsbaseret naturlig udvælgelse, for at hjælpe biologerne med at forstå hvorfor selvinteresserede individer i naturen udfører tilsyneladende altruistiske gerninger mod deres artsfæller. Med udgangspunkt i en simpel model skal der konstrueres et Java-program som giver mulighed for simulation af et antal adfærdsmønstre og deres interaktion i en biotop. I det følgende forklares den biologiske model, som forskerne ønsker implementeret, og det er jeres opgave at diskutere og dokumentere et objekt-orienteret design samt derefter at implementere og afprøve jeres simulationsprogram i Java.

Indhold

1 Formål **1**

2 Beskrivelse **2**
2.1 Modellen 3
2.2 Om brugerflade og kørsel af simulationen 5
2.3 Forslag til modelparametre 6

3 Krav til besvarelsen **7**
3.1 Krav til design 8
3.2 Krav til funktionalitet 8
3.3 Krav til afleveringen 9
3.4 Bedømmelseskriterier 10
3.5 Vigtige datoer og tidspunkter 10
3.6 Formalia 10

1 Formål

Dette er opgaveteksten til K-opgaven (karaktergivende eksamensopgave) på kurset Objektorienteret programmering og design (OOPD) i blok 2, 2006/2007. Opgaven går ud på at designe, implementere og afprøve en simpel simulation af en biotop[1] med adfærdsbetinget naturlig udvælgelse. Læs opgaveteksten omhyggeligt, specielt afsnittene med krav til besvarelsen.

[1] Wikipedia definerer en biotop som "an area of uniform environmental (physical) conditions providing habitat(s) for a specific assemblage of plants and animals [...]"

2 Beskrivelse

Biologer har gennem tiden diskuteret forskellen på og samspillet mellem gruppeadfærd og individuel adfærd. Specielt kan det undre at individer sommetider synes at ofre sig selv for andres velbefindende, når man umiddelbart skulle tro at en egoistisk adfærd ville give individet flere ressourcer til det selv. Ideen om gruppeadfærd — altså at individer frivilligt ofrer sig for flokken — er altså som antydet noget utilfredsstillende, og derfor ønsker vi i denne opgave at undersøge, hvordan man udelukkende ud fra individets selv-interesserede perspektiv kan forklare at gensæt, der koder for altruistisk adfærd, kan komme til at dominere i populationen.

Lad os forestille os en art af finker[2]. Disse finker er i den uheldige situation at tæger ofte sætter sig i deres fjerdragt. Hvis tægerne får lov at sidde længe i fjerdragten, medfører det typisk en infektion, der fører til finkens død. For at undgå dette hjælper finkerne parvis med at fjerne tæger fra hinandens fjerdragt, de steder hvor de ikke selv kan nå med næbbet. Denne altruistiske adfærd kan give anledning til undren, for hvis en finke udviste en egoistisk adfærd og modtog hjælp men aldrig gav hjælp, ville den have bedre tid til at søge føde, passe sine unger og forsvare sig mod trusler. Hvis der derfor gennem mutation eller genetisk udveksling opstod et gensæt, der kodede for egoistisk adfærd, skulle man tro at dette gensæt gennem naturlig udvælgelse over tid ville dominere flokken[3]. Hvis vi forestiller os to finker stillet overfor hinanden, kan de hver især vælge at hjælpe den anden eller at undlade at hjælpe. Dette er vist i følgende tabel med en pointsætning af hvor godt udfaldet er for den ene finke:

	Hvad du gør...	
Hvad jeg gør...	*Hjælper*	*Undlader*
Hjælper	Jeg får hjælp, men bruger tid på dig (3 pt)	Får ikke hjælp og bruger tid på dig (0 pt)
Undlader	Jeg får hjælp og bruger ikke tid på dig (5 pt)	Får ikke hjælp og bruger ikke tid på dig (1 pt)

Fra en finkes synspunkt er *Undlad* den bedste strategi, fordi den herved er bedre stillet end ved at hjælpe — uanset hvad modparten vælger at gøre.

Den ovenstående tabel beskriver det strategiske spil, der er kendt som *Prisoner's dilemma*[4]. Prisoner's dilemma er blot et af utallige problemer, som man studerer inden for *spilteori* (en: *game theory*). Spilteori er en gren af anvendt matematik og økonomi, hvor man studerer interaktionen mellem selv-interesserede, nytteoptimerende, autonome agenter[5].

Tabel 1 To finkernes udbytte i livspoint.

Finke 1 \ Finke 2	Hjælp	Undlad
Hjælp	3 \ 3	0 \ 5
Undlad	5 \ 0	1 \ 1

Når man betragter begge finkers udbytte side om side (Tabel 1), opdager man en interessant egenskab ved spillet: Den enkelte finke er som sagt bedst stillet ved at vælge *Undlad*, men samtidig er det klart

[2]Ordet finke kan måske antyde en stærkere forbindelse til Darwins forskning end egentlig er tilfældet. Denne opgave beskæftiger sig kun med adfærdsbetinget selektion og beskriver således kun et ganske lille hjørne af den moderne evolutionsteori.

[3]Det ville naturligvis betyde flokkens uddøelse, hvis alle individer var egoistiske. Man kan i lyset af denne diskussion overveje, hvorfor det ikke er sket.

[4]Man kalder spillet for Prisoner's Dilemma, fordi en af mange mulige anvendelser er situationen, hvor to tilfangetagne medkonspiratorer sidder i hver deres celle og spekulerer over om de skal tilstå overfor politiet eller holde tæt. Ved at være den eneste der tilstår får man straf-fritagelse uden senere risiko at blive genanklaget. Tilstår man ikke, risikerer man at modparten i stedet tilstår. Prisoner's Dilemma er et overraskende dybt strategisk spil, hvilket blandt andet kan ses af at der er skrevet adskillige bøger kun om det.

[5]Spilteori er blevet anvendt i biologi, datalogi, filosofi, økonomi, politik, psykologi, og sociologi til analyse af blandt andet kernevåbenkapløb, auktionsmekanismer, frie markeder, forbrugeradfærd og peer-to-peer netværk. I populærkulturen beskriver filmen *A Beautiful Mind* spilteoretikeren John Nashs forskning, som blandt andet gav anledning til begrebet *Nash-ækvilibrium*.

for begge finker, at en rationel modpart også vil spille *Undlad*. Det betyder at begge finker får 1 point. Havde begge finker i stedet spillet *Hjælp*, kunne de begge have fået 3 point, men dette resultat afhænger af at de stoler på hinanden. Spiller man kun én runde af Prisoner's dilemma, er den rationelle strategi derfor *Undlad*.

I en biotop vil det dog være naturligt at finkerne lever længe nok til at møde hinanden igen og igen, og derfor er en enkelt runde af spillet ikke tilstrækkelig. Som en model for finkernes adfærdsmuligheder over tid, bruger vi derfor et (ikke på forhånd fastsat) antal gentagelser af spillet. Det giver os mulighed for at undersøge hvilke langsigtede strategier, der giver højst udbytte. Her er det ikke længere åbenlyst, hvad en god strategi er, idet den strategi, der giver højest udbytte, vil afhænge af modpartens strategi og *vice versa*. I vores biotop betyder udbytte overlevelse og dermed mulighed for at sætte afkom med samme adfærd i verden.

2.1 Modellen

For at undersøge det beskrevne problem modelleres biotopen som et to-dimensionalt, rektangulært bræt, hvis kanter er parvis forbundet (man kan eventuelt forestille sig at finkerne lever på overfladen af en torus). Modellen startes ved at placere et antal individer på dette bræt og udføres derefter trinvist. Efter tilstrækkeligt mange trin, vil visse typer adfærd være uddøet, mens andre vil dominere populationen af finker.

2.1.1 Biotopen

Biotopen kan ses som et $n \times m$ spillebræt, hvor hvert felt kan rumme højst én finke. Det er vigtigt at bemærke, at kanterne af brættet er forbundet således at fx $(i, m-1)$ og $(i, 0)$ er nabofelter. Ved et nabofelt til et felt (i, j) forstås et af felterne $((i \pm 1) \mod n, j)$, $((i+1) \mod n, (j \pm 1) \mod m)$, og $(i, (j \pm 1) \mod m)$.

2.1.2 Finkers udklækning, overlevelse og død

Hvis en finke efter lang tid ikke er blevet hjulpet, får den infektion i fjerdragten og dør[6]. Dette modelleres gennem et antal *livspoint*, som hver finke er udstyret med. Hver runde koster livspoint for en finke, men hver gang den bliver hjulpet, får den flere livspoint svarende til pointene i Tabel 1.

Finkerne kan sætte afkom i verden, så længe de lever, og gør dette med faste intervaller. En nyudklækket finke har samme adfærd som sin forælder (se dog udvidelsesmuligheden nedenfor under **Mutationssandsynlighed**). Disse betragtninger giver følgende simulationsparametre:

Ynglesandsynlighed Dette tal bestemmer sandsynligheden for at en finke sætter et nyt afkom i verden i en runde. Det kan fx være $\frac{1}{3}$ svarende til at en finke i gennemsnit får en unge hver tredje runde.

Rundepris Alle finker mister dette antal livspoint i hver runde (uanset om de hjælpes eller ej). Det afspejler den helbredsomkostning en finke betaler for noget af tiden at have tæger i fjerdragten.

Maks. livspoint En finkes livspoint kan aldrig overstige dette tal. Det svarer til at dens fjerdragt er helt ren for tæger.

Startpoint Det antal livspoint en nyudklækket finke har.

Minimumlevealder, Maksimumlevealder Når en finke udklækkes får den en levelalder målt i antal runder. Levelalderen er et tilfældigt tal mellem **Minimumlevealder** og **Maksimumlevelalder**.

Mutationssandsynlighed [Ikke-obligatorisk udvidelsesmulighed] Sandsynligheden for at en nyudklækket finke er muteret, fx ved at den får en tilfældigt valgt anden adfærd i stedet for blot at arve sin forælders adfærd.

[6] Ingen finker har lidt overlast under udarbejdelsen af dette eksamenssæt.

Når en finke udklækkes placeres den på et ledigt nabofelt til sin forælder. Har en finke ingen ledige nabofelter, kan den ikke sætte afkom i verden. Det er valgfrit om den i så fald igen må vente det faste antal runder eller om den kan udnytte muligheden for at sætte afkom så snart et frit nabofelt opstår.

2.1.3 Møde

En finke kan møde en anden finke, hvis der er en anden finke på et af dens nabofelter. Har finken flere naboer vælges en tilfældigt. I hver runde har hver finke højst et møde.

Ved et møde vælger hver finke sin adfærd (*Hjælp* eller *Undlad*), og når begge finker har valgt adfærd, tilordnes de hver deres udbytte som angivet i tabellen. En finke har naturligvis ikke mulighed for at vide hvad den anden finke vil gøre, men finkerne kan genkende hinanden og basere deres adfærd på hvordan modparten har behandlet dem tidligere.

Bemærk, at finkernes viden udelukkende er baseret på deres egne, individuelle erfaringer. Der deles ingen information på gruppeniveau, og man kan som finke kun se hvordan en modpart tidligere har behandlet en selv, ikke hvordan modparten har behandlet andre. En finke kan heller ikke se modpartens adfærdstype, men efter et antal møder kan den selvfølgelig prøve at gætte, baseret på hvad modparten har gjort ved de første møder. (En ikke-obligatorisk variant af simulationen er at lade nyudklækkede finker arve deres forældres viden om andre finker.)

2.1.4 Adfærdstyper

Nogle basale adfærdstyper er:

Samaritaner [Samaritan] Denne finke hjælper altid.

Snyder [Cheater] Denne finketype hjælper aldrig.

Tilfældig [Random] Denne finketype opfører sig tilfældigt med lige stor sandsynlighed for hver type adfærd. (En ikke-obligatorisk udvidelse er at parametrisere *Random* over sandsynligheden for at den spiller *Hjælp*.)

Nag-bærende [Grudger] Denne finke hjælper altid, indtil modparten undlader at hjælpe. Når først en modpart én gang har undladt at hjælpe, hjælper denne finke aldrig nogensinde denne modpart igen, uanset hvor mange gange modparten ellers hjælper.

Noget-for-noget [Tit for tat] Denne finke hjælper altid, når den møder en ny modpart. Herefter spiller den samme som opponenten spillede mod den selv sidste gang de mødtes.

Mere komplicerede adfærdstyper kunne være:

Mistænksom noget-for-noget [Suspicious tit for tat] Spiller *Undlad* første gang den møder en ny modpart, og spiller derefter som *Noget-for-noget*.

Dristig noget-for-noget [Probing tit for tat] Spiller som *Noget-for-noget*, men spiller med tilfældige mellemrum — fx cirka hver femte gang — *Undlad* for at indkassere den høje gevinst.

Vendekåbe [Flip-flopper] Denne finke spiller skiftevis *Hjælp* og *Undlad*, startende med *Hjælp*.

Metafinke [Meta-finch] Har en metastrategi som løbende skifter mellem flere af de ovenstående basale strategier. En mulig metastrategi forsøger at genkende hvilken strategi opponenten spiller efter og vælger så den bedst kendte modstrategi.

Design-en-finke [Design-a-finch] Konstruer selv jeres egne finker og se om de gennem deres adfærd kan komme til at dominere populationen.

Bemærk at finkerne ikke har mulighed for at bevæge sig rundt på brættet. I stedet udvider en population sig ved at sætte nyt afkom i verden på nabofelter. Hvis tiden tillader det, kan modellen gøres mere realistisk ved at implementere bevægelse blandt finkerne.

2.1.5 Modellens cyklus

Før første runde placeres et antal nyudklækkede finker tilfældigt på brættet. For eksempel kan man beslutte at lade fem adfærdstyper dyste mod hinanden og initielt indsætte 40 finker af hver — altså i alt 200 finker tilfældigt fordelt på brættet. Herefter startes simulationen.

En runde består af følgende trin:

1. Alle finker med mulighed for det, sætter nye finkeunger i verden.
2. Finkerne mødes parvis med nabofinker og tildeles gevinster svarende til udfaldet.
3. Alle finker betaler rundeprisen.
4. Alle finker med livspoint mindre eller lig 0 eller som har overskredet deres maksimumalder fjernes fra brættet.

2.1.6 Biologiske fænomener

De to mest almindelige fænomener er ækvilibrium og invasion. Ækvilibrium kan enten være at befolkningerne uddør (degenereret ækvilibrium) eller at de fylder det meste af brættet med (stort set) konstante forhold mellem de forskellige populationer. Invasion ses ved at to befolkninger er naboer og den ene befolkning begynder at ekspandere ind over den anden. Bemærk, at der kan være "venlige" invasioner i den forstand at en strategi invaderer en anden, simpelthen fordi de trives godt i hinandens selskab.

2.1.7 Baggrund

Denne opgave er baseret på kapitel 12 (*Nice guys finish first*) af bogen *The Selfish Gene* [Daw06] af Richard Dawkins. *The Selfish Gene* fokuserer blandt blandt andet på hvordan genets selv-interesse overraskende nok leder til at individer opfører sig altruistisk over for hinanden. Kapitel 12 beskriver professor Robert Axelrods kendte konkurrence, hvor spilteoretikere, politiske strateger og biologer fra hele verden blev inviteret til at indsende adfærdstyper som dystede i tre forskellige turneringsvarianter. Vi skal ikke afsløre her, hvem vinderen blev, men blot nævne, at det er en af de i opgaven beskrevne adfærdstyper (eller måske jeres?) Axelrods efterfølgende artikel *The Evolution of Cooperation* [Axe81] er i øvrigt en af de mest citerede artikler i Science nogensinde.

Mens det **ikke er nødvendigt eller pointgivende** at gå dybere ind i dette område i opgaven, kan særligt interesserede studerende måske have fornøjelse af at læse siderne *The Evolution of Cooperation*, *Prisoner's dilemma*, *Game theory*, *The Selfish Gene* på Wikipedia. Kapitel 12 fra *The Selfish Gene* er tilgængeligt til venligt udlån eller kopiering i N208.

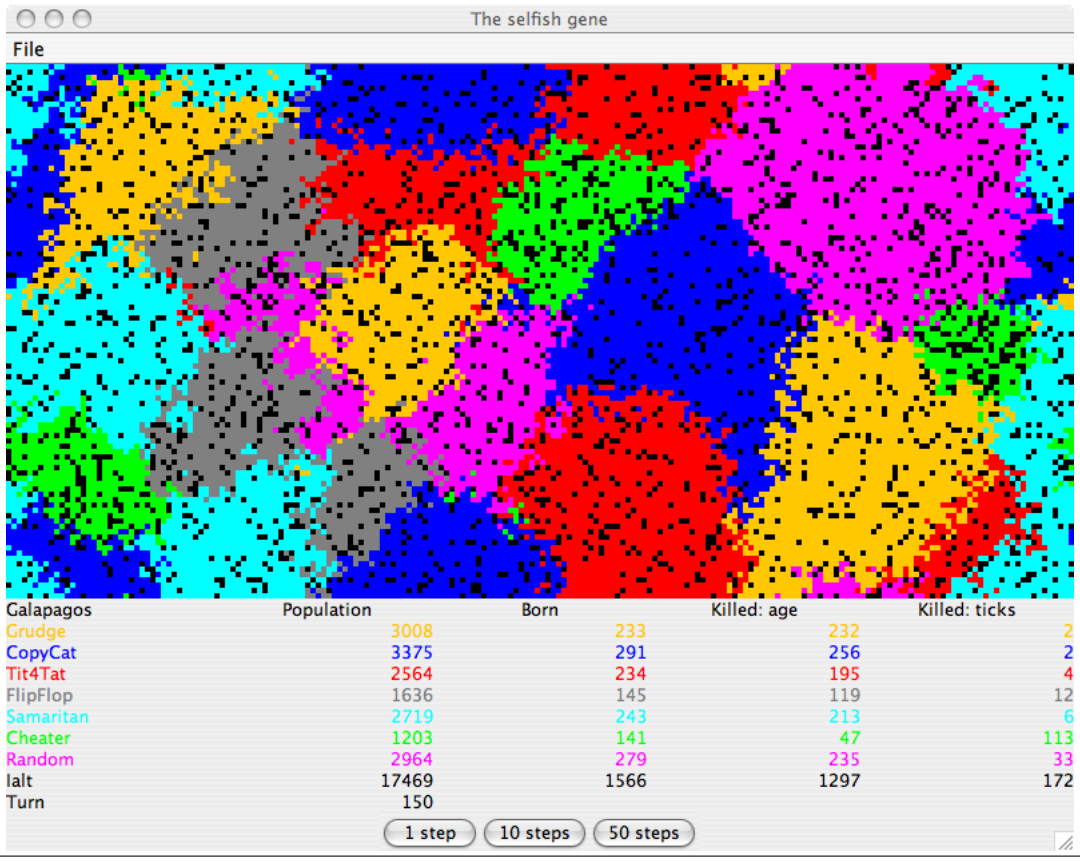
2.2 Om brugerflade og kørsel af simulationen

Simulationen udføres som skitseret i afsnit indledningen af afsnit 2.1. Brugeren interagerer med modellen ved at køre en eller flere runder ad gangen. Dette kan ske ved at brugeren trykker på en af flere knapper, der angiver et fast antal runder, der skal køres; ved at brugeren angiver antallet af runder i et tekstfelt; eller ved en kombination af disse muligheder. Det er også nyttigt, men ikke påkrævet, at give brugeren mulighed for at justere hastigheden af simulationen (fx i antal millisekunders ventetid mellem hver runde). Et (ikke-bindende) forslag til en grafisk brugerflade er vist i figur 1.

Hvis tiden tillader det, kan I endvidere tillade brugeren at foretage ændringer i finkepopulationerne i den aktuelle simulation ved for eksempel at placere nye finker på brættet med musen. Selv hvis I ikke implementerer en sådan funktionalitet, er det vigtigt, at jeres design og program ikke gør en sådan fremtidig udvidelse prohibitivt vanskelig.

Efter hvert runde rapporteres:

Figur 1 Forslag til brugerflade



1. Antal nyudklækkede og døde (skeln gerne mellem død pga. levealder og død pga. tægeinfektion).
2. Samlet befolkningstal og befolkningstal fordelt på adfærdstype.
3. Det samlede antal runder simulationen har kørt.

Selv om simulationen kører flere runder ad gangen uden stop, skal *view*'et opdateres mellem hver runde. Da dette kræver et lille særlig teknik, henvises der til side 753–754 i Nino & Hosch samt til slides 90–94 (*Delaying IndependentPlayer's move*, *Scheduling a move to delay the action*, *Delaying by timer call-back*) fra forelæsning #12 for en kodestump, som I er velkomne til at lade jer inspirere af.

Simulationens parametre kan indlæses fra en (menneske-læsbar) fil eller blot angives i `main`-metoden i klassen `Galapagos`)

2.2.1 Den udleverede klasse `AreaPanel`

Da det kan være vanskeligt at opnå tilfredsstillende grafisk ydeevne uden at bruge nogle teknikker der ligger uden for pensum, udleveres klassen `AreaPanel`, som er en Swing-komponent (underklasse af `JPanel`), der kan bruges til at vise et to-dimensionalt spillebræt. For dokumentation se filen `AreaPanel.java`, som kan hentes fra kursushjemmesiden under punktet "Eksamen" → "Regulær eksamen".

2.3 Forslag til modelparametre

Det foreslås at starte modellen på en verden der er mindst 100 × 100 felter stor med en startfinkepopulation på 1-4% af brættet. I en almindelig simulation skal man påregne af køre mindst 250 runder for at kunne observe de interessante fænomener. Som med de fleste simulationer kan det være nødvendigt at eksperimentere med parametrene for at få tilfredsstillende resultater. Prøv med følgende opsætninger:

	#1	#2	#3
Spillebræt	100×100	100×100	100×100
Ynglesandsynlighed	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{12}$
Rundepris	3	4	2
Maks. livspoint	20	20	20
Startpoint	7	10	13
Min.-levealder	10	20	11
Maks.-levealder	13	23	14
Initielt antal finker pr. strategi	40	40	40
Antal strategier	5	5	5

Man skal være uhyre forsigtig med at ændre forholdene mellem pointene i spillets udbyttematrix (Tabel 1), og de er derfor udeladt af opsætningerne. Ændrer man eksempelvis udbyttet ved ensidig hjælp fra 5 / 0 til 7 / 0, så ændres spillets egenskaber *fundamentalt*. Det er naturligvis i orden at skalere alle udbyttepoint med samme konstant, hvis man ønsker det.

Når først modellen virker, er det spændende at tilføje finker med nye adfærdstyper for at se dynamikken i befolkningerne. Hvis I finder på en adfærdstype som kan slå alle andre nævnte adfærdstyper i en-mod-en biotoper, vil vi meget gerne høre om det.

3 Krav til besvarelsen

Dette er en opgave i objektorienteret design, programmering og afprøvning, og det må derfor indskærpes, at der ikke kan gives point for yderligere biologiske eller spilteoretiske overvejelser af modellen[7]. Eventuelle uklarheder i modellen vil blive præciseret på kursets hjemmeside under "Nyheder og diskussionsforum", hvor der også kan stilles spørgsmål. Det er derfor vigtigt, at I løbende læser denne. Det kan være nødvendigt at omforme de skitserede algoritmer, så de harmonerer med jeres objektorienterede design

Det anbefales kraftig ikke at forsøge en meget ambitiøs løsning førend en simplere løsning fungerer tilfredsstillende. Det anbefales videre at konstruere modellen tilfredsstillende inden man påbegynder konstruktion af den grafiske brugerflade. *Efter man har fastlagt sit design*, kan man eksempelvis implementere efter disse milepæle:

1. Implementer og afprøv spillebrættet *i modellen*
2. Implementer og afprøv finker og deres ansvar (bl.a. udklækning, død og møde)
3. Implementer og afprøv mødet mellem to finker
4. Styk komponenterne sammen og afslut modellen
5. Konstruer en *view*-komponent, der logger aktivitet i biotopen til `System.out`.
6. Konstruer den grafiske brugerflade (først *view* sidenhen *control*)

Det er naturligvis hensigtsmæssigt at foretage løbende afprøvning.

[7] Antallet af udvidelser og yderligere overvejelser er praktisk talt ubegrænset. Vi opfordrer til at I — når I har en kørende implementering — eksperimenterer med forskellige adfærdstyper. Dette er ikke en del af eksamen, men en del af det, der gør opgaven interessant. Prøv det, men brug ikke for meget tid på det.

3.1 Krav til design

I denne opgavetekst skal en komponent forstås som en mængde af Java-klasser og -grænseflader, som tilsammen stiller sammenhængende funktionalitet til rådighed. Designet skal følge en *model-view-control* (MVC) arkitektur:

- En *model*-komponent, som indeholder biotopen. Modelkomponenten skal indeholde en klasse `Biotope`, som udvider Java-klassen `Observable`. Tilstanden af en `Biotope`-instans repræsenterer en biotop (spillebræt og finker) på et givet tidspunkt. Modelkomponenten skal stille metoder til rådighed, som gør det muligt for kontrolkomponenten (se nedenunder) at udføre et trin, der simulerer udviklingen af biotopen over tid.
  Det skal fremgå, hvorledes styringen af finkers parametre (livspoint etc.) foretages, hvorledes en finke bestemmer sin adfærd (evt. baseret på tidligere erfaringer), og hvorledes en finke deltager i møder, sætter afkom i verden og afgår ved døden.

- Der skal være en *view*-komponent, som observerer modellen:

  1. En grafisk brugergrænseflade (graphical user interface, GUI) klasse `BiotopeViewer`, som viser biotopen og finkerne deri.
  2. En logging-klasse `BiotopeLogger`, som skriver meddelelser til konsollen (`System.out`) efter hver runde, fx information om antal nyudklækkede, antal døde, samlet population og populationsfordeling.

  Begge komponenter skal implementere Java-grænsefladen `Observer` og fungere ved, at en instans af hver klasse registrerer sig på den `Biotope`-instans, der skal observeres.

- En *controller*-komponent, som styrer modellen ved at:

  1. bede modellen om at udføre et antal simuleringstrin;
  2. evt. ændre på simulationsparametre (ikke obligatorisk).

Alle komponenter må gerne indeholde andre klasser og grænseflader end de eksplicit nævnte.

3.2 Krav til funktionalitet

- Der skal være mindst 100×100 felter i biotopen og mindst de fem basale adfærdstyper.

- Programmet skal fungere, dvs.:

  1. At spillebræt og finker opfører sig som beskrevet i modellen og simulationen kan udføres.
  2. At brugerfladen virker og tillader de i afsnit 2.2 skitserede minimumskrav

- At programmet er afprøvet med unit tests (JUnit) og eventuelle fejl er beskrevet samt ledsaget af en opskrift på mulig fejlretning.

- Designet skal gøre det nemt (ved tilføjelse af nye klasser og mindre ændringer i eksisterende kode) at udvide simulationen med fx nye dyrearter, bevægelse eller forskellige felttyper på spillebrættet.

- Det skal være muligt at starte simulationen ved invokation af en `main`-metode. Simulationen skal desuden kunne kontrolles fra GUIen (*view*-komponenten).

- Den grafiske grænseflade skal være enkel. Der foretrækkes en implementering, som bruger få GUI-klasser, og som er kort, frem for en, der bruger mange GUI-klasser eller er lang.

- Det er ikke tilladt at bruge networking og reflection. Det er tilladt at bruge dele af Java 5.0, der tilhører gennemgåede dele af Java (herunder Swing, Collections og java.io), selv om disse ikke eksplicit er blevet nævnt på kurset: java.awt og pakkerne java.awt.*, java.io, java.lang, java.math, java.util, javax.swing og javax.swing.* er i orden. (Med andre ord: Man må gerne lede efter og bruge eksisterende grænseflader og klasser i disse pakker, selvom disse ikke er blevet nævnt i lærebogen.)
- Simulatoren skal kunne startes ved kald af `main`-metoden i en klasse med navnet `Galapagos`.

3.3 Krav til afleveringen

Der skal afleveres en jar-fil indeholdende programmet, dvs. alle Java-kildekodefiler med JUnit tests, og rapporten (i PDF-format). Mere specifikt:

1. En rapport med navnet `rapport.pdf` indholdende følgende dele:
   (a) Titel
   (b) Forfattere, deres adresser og eksamensnumre
   (c) Afsnit: Programdesign. Overvejelser (herunder afgrænsning), der har ført til besvarelsens design, eventuelle realistiske alternativer og deres fordele og ulemper. Beskrivelse af programdesignet. Dette kan inkludere CRC-kort, et eller flere diagram(mer) og eventuelt anvendte designmønstre i det omfang dette begrunder og illustrerer det valgte programdesign.
   (d) Afsnit: Konklusion. Da formålet med opgaven er at lave et kørende program, skal dette afsnit indeholde en kort opsummering af status, dvs. om programmet kører, og om de væsentligste fejl og mangler. Husk at man godt kan afprøve de enkelte klasser, selv om det samlede program ikke fungerer helt.

   Alle afsnit må indeholde underafsnit. Det skal bemærkes, at kvaliteten i både rapport og kildekode bedømmes i henhold til informationsdensitet og forståelighed: normalt betyder det jo kortere des bedre. Stave- og formuleringsevnen indgår i bedømmelsen.
2. Java-kildekoden (undtagen JUnit test-klasser) forsynet med JavaDoc-kommentarer[8], der for hver klasse og grænseflade rummer en beskrivelse samt pre- og postconditions.
3. JUnit test-klasser forsynet med JavaDoc-kommentarer, der beskriver hvordan man har taget specifikationen og derfra er kommet frem til afprøvningsdata (*black box testing*).

Java-kildekoden inklusive de indlejrede kommentarer skal være stillet op med passende indrykninger med henblik på læsbarhed, og der skal anvendes en systematisk og meningsfuld navngivning af klasser, variable etc.

Hver gruppe vælger en kontaktperson blandt gruppens medlemmer. Kontaktpersonen modtager korrespondance, når rapporten er evalueret og har desuden ansvar for at opbevare en fysisk sikkerhedskopi af rapporten og kildekoden. Rapporten skal uploades af kontaktpersonen til ISIS som en jar-fil på formen *fornavn1_efternavn1.fornavn2_efternavn2.fornavn3_efternavn3.fornavn4_efternavn4.jar* svarende til gruppens medlemmer. Når afleveringen uploades, er det vigtigt, at alle gruppens medlemmers fulde navn, adresse og eksamensnummer fremgår af rapporten i jar-filen.

Det er *altafgørende*, at gruppens kontaktperson i *god tid* (dvs. adskillige dage før påtænkt aflevering) gør sig fortrolig med processen inden den endelige aflevering ved for eksempel at uploade en foreløbig jar-fil eller lignende.

[8] Den fra JavaDoc genererede HTML kode skal *ikke* afleveres.

3.4 Bedømmelseskriterier

Der lægges vægt på at der afleveres et kørende program, dvs. at et simpelt og lille men veldesignet, kørende program foretrækkes frem for et ambitiøst eller stort program, der ikke rigtig virker. Det er vigtigt, at individuelle klasser er veldesignet, specificeret, implementeret og afprøvet, selv hvis programmet som helhed ikke kan køre. Det anbefales stærkt at designe, implementere og afprøve et system med meget enkel funktionalitet, inden eventuelle udvidelser føjes til.

Besvarelsen bedømmes ud fra deltagernes evne til at omsætte kursets målsætninger: at udarbejde et objekt-orienteret design og så programmere og afprøve det vha. af de på kurset gennemgåede faciliteter i Java. Se kursets målsætning på kursets hjemmeside under "Kursusbeskrivelse" og punktet "Formål" herunder. En egentlig problemanalyse indgår ikke i bedømmelsen, hverken fra et fagligt perspektiv om eksempelvis spilteori eller et biologisk-fagligt perspektiv om fx adfærdsbiologi, eftersom disse emner ikke er genstand for kurset.

3.5 Vigtige datoer og tidspunkter

Dato	Tid	Aktivitet	Sted
12/1	14:00	Eksamensopgaven stilles	Hjemmesiden på ISIS
15-18/1	13-15	Instruktorvagter	Ved den blå væg i Hel
15/1	15-16	Spørgetime v/Christian	Lille UP-1
19/1	14:00	Afleveringsfrist	Hjemmesiden på ISIS under “Aflevering – Eksamen”

3.6 Formalia

Dette er en gruppeopgave som skal løses i grupper à tre eller fire personer. Der tillades ikke genaflevering. Under helt særlige omstændigheder (sygdom, dødsfald, m.v.) kan der gives en udsættelse efter dispensation fra studienævnet. En dispensationsansøgning skal være studienævnet i hænde inden afleveringsfristens udløb. Yderligere oplysninger kan findes i eksamensbestemmelserne for Det Naturvidenskabelige Fakultet samt studieordningen for Datalogiuddannelsen. Eksamenssnyd (herunder afskrift fra andre grupper eller frihjuleri) behandles efter Københavns Universitets disciplinærbestemmelser og kan medføre annullering af samtlige eksamensresultater for alle kurser det pågældende studieår samt relegering. Eksamensopgaven udgøres af nærværende dokument samt eventuelle supplerende beskeder fra kursets undervisere, det måtte være nødvendigt at give på kursets hjemmeside enten under “Nyheder og diskussionsforum” eller “Eksamen”. Det er deltagernes ansvar løbende at holde sig orienteret på hjemmesiden.

Litteratur

[Daw06] *The Selfish Gene,* Dawkins, Richard. 30th anniversary edition, Oxford University Press, (2006)

[Axe81] *The Evolution of Cooperation*, Axelrod, Robert. Science, 211(4489):1390-6, (1981)