Utvidelse: Statistikk og lyd

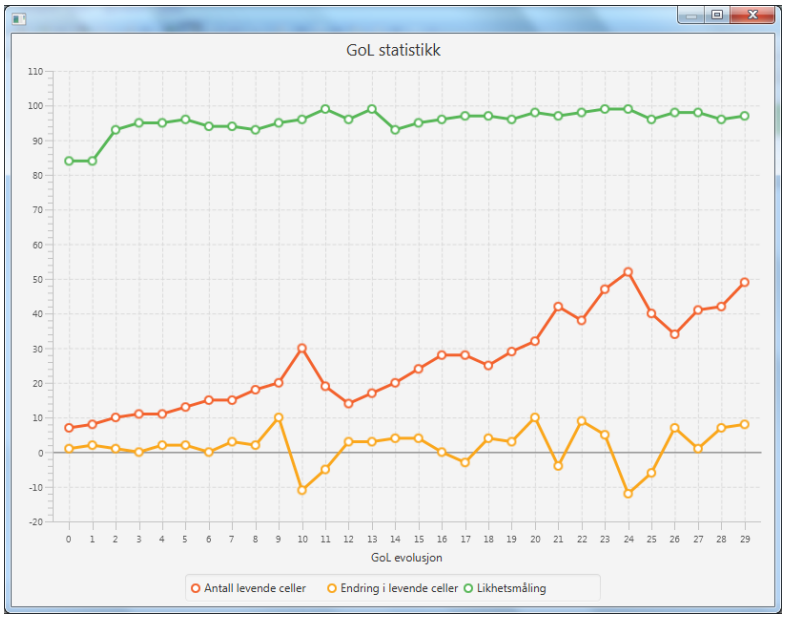
DATS1600 utvidelsesoppgave for semesteroppgaven.

Presentasjon og bruk av statistikk

Merk: denne oppgaven kan se litt matematisk ut. Dette er en litt matematisk oppgave hvis dere ønsker å utforske den mer avanserte delen på slutten av oppgaven (forbedringsoppgaven). Standard-oppgaven kan imidlertid utføres uten matematisk 'innsikt', så lenge dere forstår hva som skal implementeres. Hvis dere får problemer med å forstå oppgaven, så spør deres kontaktperson om hjelp.

I denne oppgaven ønsker vi å presentere utviklingen av mønsteret på spillebrettet i form av statistisk data. Vi kan også bruke statistikk til å la programmet gjøre forutsetninger. Slik bruk at statistikk er et sentralt konsept innen *kunstig intelligens*.

Vi kan se for oss følgende presentasjon av de tre statistiske distribusjonene dere skal implementere i denne oppgaven:



Dere skal implementere en metode som samler inn data over en angitt periode. Foreslått signatur:

```
int[][] getStatistics(GameOfLife game, int iterations) {…}
```

der den returnerte 2-dimensjonale tabellen skal returnere statistisk data over antall iterasjoner (dvs. antall kall til nextGeneration()). For eksempel, kan antall kolonner være iterations+1 (pluss 1 fordi vi vil inkludere data for mønsteret vi starter med, dvs. når tid=0). Antall rader vil være 3 for denne oppgaven fordi dere skal samle inn tre typer data.

Tenk på hvor i programstrukturen denne metoden skal plasseres. Hvis metoden plasseres i klassen (eller en arvet klasse) relatert til spillet (f.eks. GameOfLife), så er parameteren *game* unødvendig.

For å samle inn statistikk over flere generasjoner, må iterative kall til nextGeneration-metoden utføres. Vi ønsker ikke å utføre slike kall direkte på spillobjektet som vises på skjermen fordi brettet ikke skal endres etter kall til getStatistics. Spillet (eller spillebrettet) må dermed *kopieres* (deep copy)

før eventuelle kall til nextGeneration. Hvis dere ikke har implementert en slik kopieringsoperasjon, så implementer dette (se utvidelsesoppgaven for mønstermanipulering for informasjon).

I denne oppgaven skal dere samle inn følgende data for hver generasjon:

1. antall levende celler
2. endring i antall levende celler
3. likhetsmåling for nærmeste generasjon (en: similarity measure)

Antall levende celler

For hver generasjon, skal antall levende celler på spillebrettet telles. Tenk på hvor i program-strukturen denne oppgaven skal utføres.

Denne telleoppgaven skal returnere en tabell der hvert element i tabellen angir hvor mange levende celler det er i en generasjon av spillet.

Matematisk sett, er denne tabellen en *diskret funksjon* $f(t)$ som angir antall levende celler for en angitt generasjon t av spillet (t refererer til tid). Denne parameteren t er definert innenfor domenet $t \in [0, it] \subset \mathbb{Z}$, der it er antall iterasjoner brukeren har angitt. For eksempel, hvis $f(5) = 30$, så betyr det at spillet har 30 levende celler i sin 5. generasjon.

Endring i antall levende celler

En annen type data vi kan være interessert i er hvordan mønsteret endrer seg fra en generasjon til en annen. Det er flere måter å definere denne endringen. I denne oppgaven, kan vi definere den slik:

$$e(t) = f(t+1) - f(t)$$

Matematisk sett, representerer funksjonen e den *deriverte* funksjonen til f. Dvs. at $e = f'$. Merk at domenet til e er $t \in [0, it-1] \subset \mathbb{Z}$ siden $f(it+1)$ er udefinert. I praksis, kan dere ignorere tabellverdien til det siste elementet i tabellen relatert til e.

Similarity measure

Den tredje tabellen (eller raden i 2D tabellen) skal angi en sannsynlighet, angitt fra 0 til 100. Denne verdien representerer sannsynligheten for at det finnes en annen generasjon i evolusjonen til spillet som har et identisk mønster sammenliknet med mønsteret til brettet ved tid t.

Merk at vi bruker en sannsynlighetsmåling (similarity measure) og ikke en binær måling (ja/nei). Hvis to mønstre er identiske, skal målingen for disse returnere 100 (dvs. 'ja, det finnes et identisk brett for brettet ved tid t'). På den andre siden, hvis de ikke er identiske skal målingen returnere hvor like brettene er. For eksempel, vil verdien 99 angi at brettene er veldig like, men ikke identiske, mens verdien 45 angir at brettene ikke er særlig like.

En slik likhetsmåling er spesielt vanlig innen kunstig intelligens der det ofte er umulig å konkludere med 100% sikkerhet om to elementer har noe med hverandre å gjøre. Et visuelt eksempel er problemet for ansiktsgjenkjenning, der datamaskinen har som oppgave å bestemme om et ansikt i et bilde er av samme person som et ansikt i et annet bilde. I denne problemstillingen, kan for eksempel datamaskinen konkludere, med 99% sikkerhet, at to ansikter er av samme person. Maskinen kan imidlertid ikke gi en 100 % garanti for at den har korrekt.

For å muliggjøre dette ønsker vi å redusere representasjonen av brettet vårt. I vår nåværende representasjon, er brettet en 2D tabell der hver celle enten er død eller levende. Med denne representasjonen må vi iterere gjennom alle celler for hvert mønster i alle iterasjoner, noe som kan

være ressurskrevende. I denne oppgaven, vil vi redusere representasjonen av brettet vårt ned til en enkel funksjon, som vi kaller $\Phi(\mathrm{t})$; funksjonen skal returnere en flyttallsverdi (float eller double).

Som dere kan tenke dere, er det utallige måter å redusere vår originale 2D representasjon ned til et enkelt flyttall. Vi ønsker å kombinere ulike faktorer som gjør tallet så unikt så mulig for et virkårlig mønster. I denne oppgaven, skal vi bruke faktorene relatert til utviklingen til levende celler og en geometrisk faktor. Definisjonen av $\Phi(\mathrm{t})$ er:

$$\Phi(\mathrm{t}) = \alpha f(t) + \beta f'(t) + \gamma g(t)$$

der α, β, γ er skaleringsparametere som brukes til å vekte de tre faktorene. Disse parameterne justeres vanligvis av programmerer og presenteres ikke til sluttbrukeren. Det vil si at de er interne konstanter i programstrukturen. Dere kan bruke verdiene $\alpha = 0.5, \beta = 3.0, \gamma = 0.25$.

Den geometriske faktoren $g(t)$ representerer den geometriske posisjonen til mønsteret. Denne faktoren er introdusert for å ta hensyn til at et mønster ikke nødvendigvis endrer seg så mye i antall levende celler, men kan endre seg geometrisk. Glidere er det prominente eksempelet på en slik oppførsel. Konkret, er $g(t)$ definert som summen av x- og y-koordinatene til alle levende celler på brettet ved tid t.

Nå som representasjonen av brettet er redusert til $\Phi(\mathrm{t})$, kan vi enkelt sammenlikne to brett, som vil definere vårt similarity measure. Gitt to brett ved tidene t_1 og t_2, så er dette definert ved:

$$P\big(\Phi(\mathrm{t}_1) \sim \Phi(\mathrm{t}_2)\big) = \frac{\min\big(\Phi(t_1), \Phi(t_2)\big)}{\max\big(\Phi(t_1), \Phi(t_2)\big)}$$

Math biblioteket i Java kan brukes til å evaluere funksjonene min og max.

Siden vi arbeider med int verdier (husk: returverdien til vår funksjon er int[][]), så skalerer vi sannsynlighetsverdiene til prosenter, der vi bruker konservativ avrunding:

$$P\big(\Phi(\mathrm{t}_1) \sim \Phi(\mathrm{t}_2)\big)_{\%} = floor\big(P\big(\Phi(\mathrm{t}_1) \sim \Phi(\mathrm{t}_2)\big) * 100\big)$$

Igjen kan Math biblioteket brukes for funksjonen floor (double-til-int casting kan eventuelt brukes direkte). Vi er konservative i denne avrundingen for å ikke konvertere en usikker sammenlikning til en sikker en. For eksempel, hvis sannsynlighetsverdien er 0.999, så betyr dette at mønstrene er veldig like, men ikke identiske. Heltallsverdien, i prosent, for denne sannsynligheten burde da være 99% og ikke 100%.

Vi ønsker nå å bruke vårt similarity measure til å identifisere det brettet som er mest likt brettet ved en gitt tid t. Opprett en tabell s med samme lengde som f og utfør denne oppgaven. Verdien $s(t)$ skal være den største sannsynlighetsverdien (i prosent) til t. Husk å ikke sammenlign et brett med seg selv (dvs. når $t = t_2$). Matematisk, skrives denne funksjonen slik:

$$s(t) = \max_{t_2 \in [0, it-1] \subset \mathbb{Z} \cap \neg t} P\big(\Phi(\mathrm{t}) \sim \Phi(\mathrm{t}_2)\big)_{\%}$$

Merk at dere kan endre på definisjonen til s avhengig av hva brukeren ønsker. For eksempel, kan brukeren velge en spesifikk iterasjon t_s og s kan da returnere likheten til iterasjonen t sammenliknet med t_s (dvs. $t_2 = t_s$).

Presentasjon av statistiske data

Diskusjonen over beskriver innholdet i en 2D tabell av typen int[][]. Vi ønsker å presentere denne tabellen visuelt. Les om JavaFX sin støtte for datavisualisering:

JavaFX støtte: http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/charts.htm#JFXUI577

Konkret eksempel: http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/line-chart.htm#CIHGBCFI

Hvis dere opplever at programmet fryser mens det kalkulerer de statistiske dataene (f.eks. ved store brett og mange iterasjoner), så kan det være naturlig å bruke JavaFX sin ProgressBar:

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ProgressBar.html

Bruk av statistikk

En applikasjon av $P(\Phi(t_1) \sim \Phi(t_2))$, dvs. similarity measure, er et konsept som kalles 'video textures'. En video texture er en video som aldri stopper ved å hoppe til bilder med høy 'likhet' i bildesekvensen (analogi: instruksjonshopp). Implementer en liknende funksjonalitet for GoL: istedenfor å gå til neste generasjon, gå til en generasjon med et mønster som likner mønsteret i den nåværende generasjonen. Bonuspoeng angis for gode resultater, som rapporteres via GIF filer eller video.

En variabel som kan angis av bruker er en tilfeldighetsvariabel (mellom 0 og 1), som kan genereres av et Random objekt. Hvis variabelen er større enn en angitt verdi (f.eks. 0.5), så skal spillet hoppe til en liknende generasjon. Hvis ikke, så skal spillet gå som normalt (dvs. neste generasjon med den vanlige nextGeneration metoden). Merk, at en liknende iterasjon kan være en iterasjon som allerede har blitt utført, noe som betyr at spillet hopper tilbake i 'tid'.

Original artikkel: http://www.cc.gatech.edu/cpl/projects/videotexture/SIGGRAPH2000/index.htm

Mer moderne artikkel som viser konseptet bedre: https://www.youtube.com/watch?v=drKmS-Czu44&feature=youtu.be

Forbedring

Definisjonen av $\Phi(t)$ kan forbedres og kan dermed endres, spesielt definisjonen av den geometriske faktoren (dere bør tenke på hva som er feil med den nåværende definisjonen og komme opp med mønstre som 'lurer' funksjonen til å si at to mønstre er like, selv om de ikke er det). Hvis dere kommer opp med en alternativ definisjon av $\Phi(t)$ som gir bedre resultater sammenliknet med dens originale definisjon, vil bonuspoeng angis. Dere må demonstrere, med konkrete eksempler, at deres definisjon er bedre. Presenter deres alternativ definisjon, med visuelle resultater og illustrasjoner, i en PDF fil.

Konvertering til lyd

I denne oppgaven skal dere representere spillebrettet via lyd, og ikke via en grafisk presentasjon.

Intro

Før dere starter denne oppgaven, kan det være hensiktsmessig å forstå hva lyd og lydbølger er. Kort sagt, propagerer lyd i bølger gjennom rommet. Ulike frekvenser på disse bølgene produserer ulike oppfatninger (eng: perceptions) av lyd.

Lyder i den virkelige verden er full av 'støy'. Dvs. for å representere generelle lyder matematisk (f.eks. via en parametrisk funksjon), trenger man mange frie variabler.

I denne oppgaven ønsker vi å produsere lyd (eng: synthesise). For slik produksjon, ønsker vi en enkel representasjon av lyd, som samtidig er fleksibel nok til å produsere interessante lyder. En enkel representasjon for en bølge er via sinus funksjonen, der vi kan assosiere en konstant, c, for å manipulere frekvensen til bølgen:

$$f(x) = \sin(cx)$$

Lyder på denne formen representerer 'rene' lyder, som for eksempel kan korrespondere med en piano tone. Noen eksempler på verdier av c som tilsvarer piano toner:

- A4: $c = 2 * \pi * 440$
- B4: $c = 2 * \pi * 493$
- C4: $c = 2 * \pi * 262$
- D4: $c = 2 * \pi * 294$

Bruk et visualiseringsverktøy til å visualisere sinus kurvene. Hvis du ikke bruker et visualiserings-verktøy, kan du bruke nettjenesten www.wolframalpha.com (f.eks. 'søk' på 'sin(2*pi*440*x)').

Når man kombinerer ulike toner produserer man akkorder. Matematisk, er en akkord summen av flere toner:

$$A4 + C4 \rightarrow f(x) = \sin(2 * \pi * 440x) + \sin(2 * \pi * 262x)$$

Når det gjelder lyd og musikk, kan man komme 'langt' med et lite antall akkorder. Dette er noe dere kan eksperimentere med i denne oppgaven. Eksempel:

https://www.youtube.com/watch?v=X1ynXZ_ECIs

Når man produserer digital lyd, må man påse at man bruker en angitt frekvens (eng: sampling frequency). Dette tilsvarer hvor mange punkter som vil representere bølgen per sekund. En standard frekvens, som dere kan bruke, er 44 100 prøver per sekund (dvs. et 44.1 kHz digitalt signal). I tillegg, brukes flere kanaler (channels) for å kombinere akkorder. Disse grunnleggende konseptene burde leses om i dokumentasjonen til biblioteket dere skal bruke.

I denne oppgaven skal dere bruke biblioteket Java Sound API (en del av standardbiblioteket til Java). Dette vil være en øvelse i å sette dere inn, på egenhånd, i et relativt avansert bibliotek. Det anbefales at dere skumleser offentlig dokumentasjon og eksempler før dere går videre med oppgaven. Husk: dere skal hovedsakelig produsere (synthesise) lyd og ikke ta opp (capture) lyd.

Lyd-basert representasjon av et spillebrett

Tenk på ulike måter å representere spillebrettet via et 1D lydsignal. For eksempel, kan dere iterere igjennom alle cellene til brettet og for hver levende celle produsere en lyd basert på en verdi (som antall levende naboceller). Dere kan også representere evolusjonen til spillet over tid via lyd ved å bruke statistikken til hver iterasjon.

En fremgangsmåte er å definere noen forhåndsdefinerte lyder, som for eksempel kombinasjoner av piano akkorder, der en gitt tilstand vil produsere en gitt lyd. Java Sound tilbyr også noen forhåndsdefinerte instrumenter som kan brukes.

Alternativt, kan dere hente lyder fra Internett, der dere kan assosiere en lyd med en gitt tilstand. En ressurs: http://ibeat.org/

Kombinere statistikk og lyd

For å forbedre brukeropplevelsen, kan det være naturlig å støtte avspilling av musikk (f.eks. fra WAV filer) i applikasjonen. En enkel funksjonalitet er å la brukeren velge egne sanger/filer som skal spilles av mens spillet utføres.

Et mer avansert valg er å automatisk spille av en sang eller lyd fra en forhåndsdefinert spilleliste. Ved å analysere statistikken til spillet, kan det velges en passende sang. For eksempel, hvis spillet har mye

aktivitet i seg (høyt antall levende celler eller store endringer i antall levende celler), kan en mer dramatisk sang eller lyd spilles av. Lyder kan kontinuerlig endres i løpet av utførelsen av spillet, for eksempel ved å se på statistikken til de neste 20 generasjonene, eller en lydsekvens kan velges på forhånd ved å analysere statistikken til valgt mønster over en lengre periode (som for eksempel 1 000 generasjoner).