

Assignment 3

Klement/cgx702

Forord

Min gruppe bestod af Lucas Marshall (dqr746), Noah Skovborg (tzq775), Oliver Nygaard (lqn482) og Gustav Mollerup (zjm580).

Gustav, Oliver og jeg, startede med at udvikle måden de forskellige personer skulle agere på og selve strukturen bag det. Gustav skrev, så den første udgave af Normal, Copycat og Skeptic klasserne og udvidelse af den abstrakte klasse.

Funktionen `simulateMeeting` blev udarbejdet i fællesskab, med mig som hovedforfatter.

Funktionen `createEdgeMatrix` blev udviklet i sammenarbejde på tværs af hele gruppen.

Funktionen `createGraph` blev udviklet i fællesskab med Lucas som hovedforfatter. Derudover stod Lucas også for opdatering af cirkel-tegningsfunktionerne, baseret på kode fra forelæsninger.

Oliver var hovedforfatter på funktionerne der står for at tegne kanterne på grafen, samt at få programmet til at virke med `interact`. Dette omfattede funktionerne `buildPicture`, `circleData`, `makeEdgeLine`, `edgeLines` og `react`.

Jeg foretog senere en revision, hvor der bl.a. blev tilføjet `drawPoliticalViews` funktionen og koden blev gjort mere læslig. Jeg tilføjede også `runSimulation` funktionen, som samlede de en masse gode for at bruge `canvas interact` funktionen.

Noah har så senere tilføjet en funktion, `logLinesTree`, så man kan se hvad den seneste `interaction` mellem to personer var.

Jeg har selv tilføjet `showRunde`, som viser hvor mange gange simulationen har kørt, og lavet lidt forandring i udregningerne, ved `onInteraction` funktionen.

Som kilder har vi brugt dokumentationen og eksempler fra Diku Canvas og `simpleGraph`, til at finde funktioner vi skulle bruge til at løse opgaven. Vi har også brugt `Fsharp.Core` til, at bruge higher order funktioner som `List.fold`, `List.randomSample` og `List.map`. Vi har også brugt slides fra undervisningen.

Introduktion

I opgaven skal vi lave en simulering af hvordan folks politiske holdning kunne forandre sig, når to folk mødes, baseret på en række faktorer. I denne simulering mødes folk i tilfældigt trukket par, hver runde og der udregnes en værdi baseret på hvor god match de er og andre faktorer. Vi skulle derudover også lave en graph, der repræsenterer alle participants og deres political view iforhold til hinanden. Vi skulle også tegne en visualisering af den graf.

Programbeskrivelse

Hele programmet er baseret på den Abstract Class `Participant`, som har 5 egenskaber. 4 af egenskaberne bliver givet af opgave stillingen og den inkludere kode, `Id`, `name`, `politicalView` og `influenciable` og 1 egenskab vi selv har tilføjet senere, `charisma`.

`Id` er et unikt ikke negativt tal for hver `Participant`. Vi har gjort det ved at bruge et static variable `latestID`. Det betyder at for hvert ny `Participants`, som bliver lavet, sætter vi deres id til `latestID + 1` og gemmer den værdi, som `latestID`, så vi kan sætte den næste `Participants` id til `latestID + 1` igen.

`Name` er blot en string, som vi giver når vi definere en `Participant`, vi bruger den ikke til noget konkret i programmet.

Egenskaben `politicalView` er en repræsentation af hvor `Participanten` ligger på et eller andet slags politisk spektrum. den er begrænset til at være mellem 0 og 1. Det tjekker vi mest af alt, når vi sætter `politicalView` med den indbyggede `set`, så tjekker vi om den værdi vi vil sætte den til er større eller

mindre end 1 eller 0, og sætter den så bare til 0 eller 1, hvis den er.

De sidste to egenskaber er influenciable og charisma de indikere henholdsvis, hvor meget en person bliver påvirket af andre og hvor meget de påvirker andre.

Participant har også to funktioner OnInteraction og MatchScore.

MatchScore er den samme funktion, for alle børnene vi definere, den returnere blot afstanden mellem de to participants politicalViews.

OnInteraction er den funktion, som gør den største forskel mellem børnene. Participant har en On-interaction funktion der er defineret ved at kigge på personen influenciable og charisma og afstanden mellem de to personer. Som vi regner den ud i funktionen kan den også skrives, som to funktion:

$$M(d) = a \cdot d^2 + d$$

$$a(I, c) = I \cdot c - 1$$

d =
afstanden mellem
personerne

I = influenciable

c = anden persons
charisma

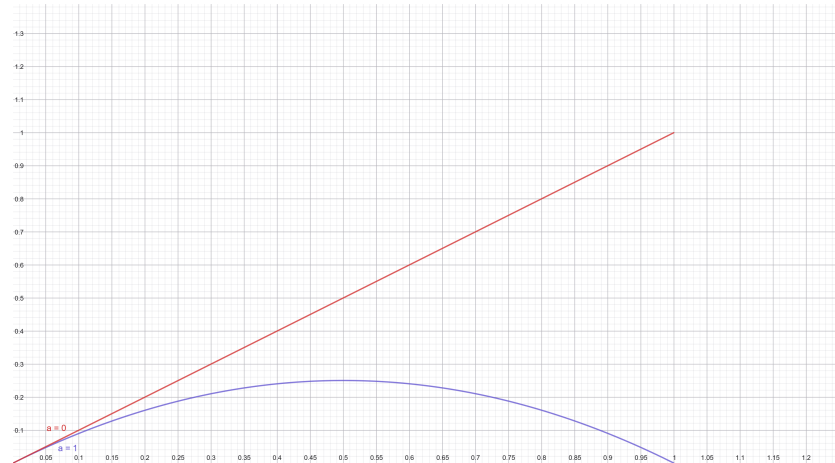
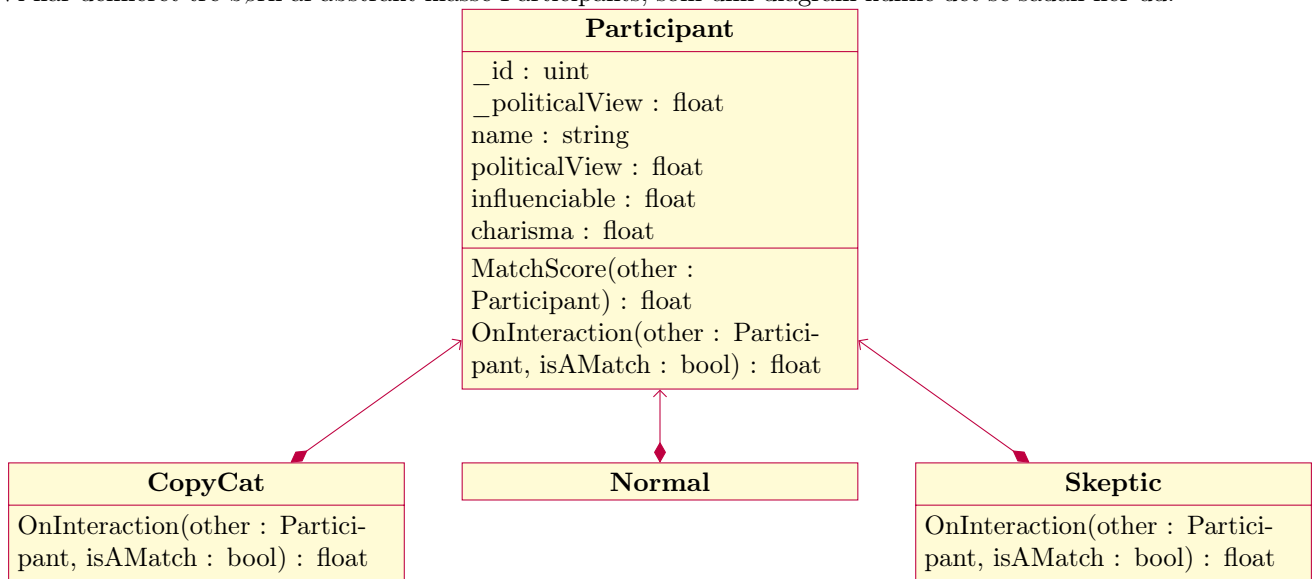


Figure 1: Graf for $a = -1$ og $a = 0$

Det er sådan set en funktion af tre variabler, men det er lettest bare at tænke på som et andengradspolynomium, hvor personerne der mødes når et punkt hvor de er så uenige at de begynder, at påvirke hinanden mindre. Og personer tæt på hinanden vil også påvirke hinanden meget lidt. Grafen viser det for $a = -1$, altså når to personer, en med 0.0 influenciable og en med 0.0 Charisma mødes. og $a = 0$, altså når to personer, en med 1.0 influenciable og en med 1.0 Charisma mødes. Det er meningen, at normal skal være et godt bud på hvordan den gennemsnitlige person bevæger sig på det politiske spektrum, de kan godt rykke sig, hvis de møder personer der er overbevisende nok.

Vi har defineret tre børn af abstrakt klasse Participants, som uml diagram kunne det se sådan her ud:



Normal barnet arver alt fra Participant uden at ændre noget. Mens Copycat og Skeptic arver alt andet end OnInteraction funktionen. De vil side at i mødet med andre måder agere de på en anden måde end den såkaldte normale person. Det valg har vi som gruppe truffet, at OnInteraction funktionen

skulle være anderledes for de forskellige typer.

CopyCat er bare en person, som er meget let at overbevise. CopyCats OnInteraction funktion betyder, at de altid bevæger sig hen til den person, som de møder lige meget hvor uenige de er, det er dog stadig relevant for dem hvad den anden persons charisma og deres egen influentiability er. Men de vil altid bevæge sig mod dem.

Skeptic er en overdrevent version af en person der stiller spørgsmålstegn ved alt. Det betyder, så at vi har valgt at lave Skeptics OnInteraction funktion sådan at de altid vil bevæge sig væk fra de personer de møder lige meget hvor tætte de er i politisk holdning. De er også stadig påvirket af influentiability og charisma, men det ender med at have den modsatte effekt. De vil bevæge sig mere væk fra en person med høj charisma end en person med lav charisma.

Vi har også defineret en ny type som vi kalder simulationState. Den indeholder vores liste over Participants, en Log, som er en string list over de seneste møder mellem Participants og Runde, som bare er antallet af runder, som vi har kørt. De to variabler er bare for egen sjov skyld og vi displayer dem i vinduet, når vi kører funktionen.

Den funktion som står for at simulere mødet mellem to personer er simulateMeeting. Den tager en Participant List, som input og den outputter, de to personer der mødte og om de var en Match. Deet er mest for loggen, at vi outputter det. I selve funktionen møder de to personer, vi udregner deres politicalMovement for hver person, baseret på onInteraction funktionen. Det betyder, at vi sådan set er ligeglad med den anden persons klasse, da det er op til den individuelle onInteraction funktion hvordan man regner det ud. Vi ændre kun værdier efter vi har regnet det ud for dem begge to, ellers ville person1 få lov til at ændre på den anden persons politicalView, før vi regner deres OnInteraction ud.

Funktionen edgeMatrix laver en liste, som vi så kan tegne kanterne ud fra. Det gør den ved at gå alle mulige par igennem og tilføje dem til listen, sammen med en bool der afgøre om de var et match eller ej. Det betyder længden af den liste altid vil være participantList.Length!. Funktionen returner så en Array med alle de tuples der indeholder den information, så vi senere kan tegne den med de rigtige kanter.

Funktionen BuildPicture tegner og samler alle de forskellige PrimitiveTrees, så draw funktionen kan bruge den. Den starter med at definere fonten, koden for det er fundet i Diku Canvas dokumentation. Vi har også antallet af cirkler, som er antallet af Participants i vores simulation. Den tager et midtpunkt i skærmen og simulationState, som input og outputter et Canvas.Picture.

Vi bruger så en funktion der hedder circleData, som returner en liste med personerne, midten af deres circle og en circle PrimitiveTree. Circle PrimitiveTree bliver lavet af funktionerne, makeCircle, line, pointPolar. Det er alt sammen kode fra undervisningen.¹ Vi ændre farven på cirklen baseret på personens politicalView. Hvis den er større end 0.5, så er de blå ellers er de røde.

Når vi så skal tegne kanterne mellem cirklerne på vores graf bruger vi funktionen makeEdgeLine, den tager to inputs, midten af de to cirkler den skal tegne imellem. Vi bruger pythagoras til at finde afstanden mellem to cirkler, for at tjekke at den ikke er 0. Hvis afstanden ikke er 0, så tegner bruger vi canvas.piecewiseAffine til at tegne en linje mellem de to cirkler, dog forskudt med en ratio baseret på cirkelens radius.

react funktionen er den der står for TimerTick og bliver kaldt af interact. For hvert TimerTick, så kører vi simulationen en gang og gemmer resultatet til log, vi returnere, så den samme state med den nye log og Runde + 1.

Afprøvning

Når jeg skal teste koden, skal jeg se om de tre forskellige klasser opfører sig på den måde som jeg forventer.

Jeg har lavet en funktion testTypes, i meetUpExamples.fsx, der tester hvordan de 3 forskellige klasser interagerer med hinanden.

Funktionen har 3 forskellige folk med hver deres klasse, Normal, CopyCat og Skeptic. Jeg simulerer et møde med hver af dem inklusiv sig selv. Jeg har sat charisma og influenciability til det samme for alle personerne, så jeg kan teste personerne opførelse i forhold til hinanden. Jeg ændrer heller ikke på værdierne, så jeg simulere ikke et møde hvor deres holdning ændres, jeg kigger bare hvilke tal de

¹J.Sporring "DIKU-Canvas og Abstrakte Datatyper", url: https://absalon.ku.dk/courses/85607/files/10111664?module_item_id=2869668 slide 16

ville være ændret med hvis jeg havde ændret dem. Jeg kigger bagefter på om de tal den returnere stemmer overens med, det forventede resultat. Det betyder at Normal person, bevæger sig tættere på begge, men lidt tættere på den person der i forvejen ligger tættere på i forhold til den person der ligger længere væk. CopyCat bevæger sig tættere på begge parter, med lige meget, fordi de begge er lige langt væk fra CopyCat. Og Skeptic bevæger altid væk fra de to andre, men længere væk fra personen de er mest uenig med.

Når vi kører testFunktionen, så giver det også det som vi havde forventet

Analyse

Når man kalder runSimulation funktionen fra modulet MeetUp giver en graf der kunne se sådan ud: Man kan også manuelt kalde simulateMeeting, hvis man ikke vil have visualiseringen. I meetUpExam-

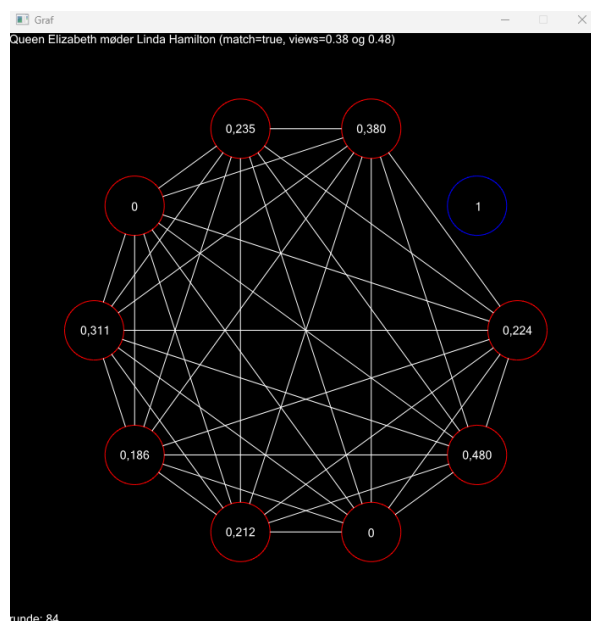


Figure 2: Graf outputtet af program

ples.fsx har jeg lavet en funktion der hedder SimulationWithNoVisual, som man kan kalde, der kører simulationen 100000 gange og printer politicalView før og efter og gennemsnittet af alle politicalViews før og efter. Hvis man kører den et par gange kan man se, at der opstår et mønster.

Jesus Christus, Elizabeth Turner og Charlie Kirk, ender altid ude på ekstremerne. Charlie Kirk på 1.0 og modsat Jesus Christus på 0.0. De er alle sammen af Skeptic klassen, og på grund af måden Skeptic er lavet på, vil de altid bevæge sig væk fra alle. Siden Charlie Kirk og Jesus Christus starter ude ved ekstremerne er det ikke overraskende, at skeptic klassen tvinger dem til at blive derude, siden de kun kan bevæge sig væk fra andre folk. Der er ikke nogle der kan være mere ekstreme og så skubbe dem ind mod midten.

Elizabeth Turner er dog ikke altid 1.0 eller 0.0. Hun starter i midten med en politicalView på 0.5, så hun ender ude på ekstremerne, hvor lang tid der går og hvilken ekstrem hun ender på kommer an på hvilke folk hun møder. Hvis hun kun møder Jesus vil hun bevæge sig hurtigt til 0.0, men hvis hun kun møder Charlie Kirk vil hun hurtigt bevæge sig til 1.0

Resten af Participants, både CopyCat plejer at ligge sig rundt om gennemsnittet, altså ± 0.15 . Dog er det ikke et fast punkt, man kan blive ved med at køre simulationen, vil gennemsnittet blive ved med at ændre sig. Gennemsnittet ligger dog aldrig omkring 0.5, men lidt op ad fløjene. Det har nok nået med hvilken side Elizabeth Turner har lagt sig ud til. Det vil altså sige at der ikke er noget sted den konvergere mod, men at den politiske verden vi simulere bliver ved med at ændre sig.

Hvis man kører en liste med kun Normals vil de dog altid finde et sted de kan være enige om, som så er den politicalView de allesammen ender med at få.

Resumé

Kildefortegnelse

- [1] F# Foundation, “ListModule – F# Core Library,” hentet fra: <https://fsharp.github.io/fsharp-core-docs/reference/fsharp-collections-listmodule.html>, tilgået 6. december 2025.
- [2] DIKU, “DIKU-Canvas kildekode (canvas.fsi),” hentet fra GitHub: <https://github.com/diku-dk/diku-canvas/blob/main/canvas.fsi>, tilgået 6. december 2025.
- [3] J. Sparring, “DIKU-Graph kildekode,” hentet fra GitHub: <https://github.com/sparring/diku-graph>, tilgået 6. december 2025.
- [4] J. Sparring, "DIKU-Canvas og Abstrakte Datatyper", hentet fra Absalon: <https://absalon.ku.dk/courses/85607/files/10111664>, tilgået 6. december 2025.