Algoritmer Unplugged

Et analogt forløb i Programmering B

Maya Saietz
H.C. Ørsted Gymnasiet,
Frederiksberg
ms@tec.dk

Resumé

I denne opgave har jeg undersøgt brugen af analoge aktiviteter med spilelementer, inspireret af Unplugged Computing, i et algoritmeforløb i en 2.g-klasse, hvor eleverne skulle øve sig på at bruge allerede kendte kontrolstrukturer til at løse nye problemer. Jeg har udviklet to sådanne aktiviteter til formålet, og bygget et undervisningsforløb op omkring dem. Jeg har derefter analyseret forløbet ud fra David Kolbs teori om erfaringslæring, og konkluderet at denne type aktiviteter har stort potentiale for at understøtte elevernes overgang fra at læse og forstå kode, til selv at kunne skrive den, og at aktiviteterne desuden kan give en fælles referenceramme for hvordan man arbejder systematisk med at udvikle og implementere algoritmer.

Indhold

1 Indledning

Ifølge vejledningen til læreplanen i Programmering B, skal programmeringsfaget ikke kun handle om at programmere, men også *'at kunne tænke selv, kunne samarbejde omkring idéudvikling, skabelse og videndeling'*[13]. Faget skal bidrage til elevernes digitale kompetencer og studiekompetencer – det fortolker jeg til dels som at faget skal forberede eleverne på at kunne begå sig i en verden der i stigende grad er digital, ikke kun som forbrugere af andres digitale produkter, men som skabere af deres egne.

I selve læreplanen står der under de faglige mål, at eleverne skal være i stand til at *'bruge programmering til at undersøge et emne eller problemområde, med henblik på — via programmets funktion – at skabe ny indsigt eller til at løse et problem'*[13]. Det er altså ikke nok at eleverne forstår hvordan de forskellige kontrolstrukturer virker, de skal også kunne finde ud af hvornår disse skal bruges.

I IT-fagenes didaktik arbejder vi meget med en *use-modify-create*-progression, hvor eleverne først skal eksperimentere med et udleveret program, derefter lave modifikationer til det, og til sidst selv udarbejde ligenende programmer[13]. Min erfaring er dog at der er et stort spring fra *use* til *modify* og *create* – det er langt nemmere at læse kode som nogle andre har skrevet, end det er at skrive sit eget, selv hvis man har en god forståelse for hvordan programmer evalueres.

En del af udfordringen kommer af at det ikke er nok at forstå de forskellige kontrolstrukturer og datatyper man har til rådighed. Der er også et kreativt arbejde i at finde på en algoritme der løser lige præcis det problem man sidder med. Dette indvolverer at sætte sig selv i computerens sted og forstå hvordan man kan løse problemet ved hjælp af en begrænset mængde af meget simple operationer, og derefter at oversætte sin forståelse til et konkret programmeringssprog. Erfarne programmører har set mange forskellige eksempler på dette, og ved præcis hvilke egenskaber de skal lede efter når de undersøger deres problem. Denne erfaring kan eleverne kun delvist nå at opbygge i løbet af de to års undervisning.

Formålet med dette projekt er at give eleverne nogle værktøjer til at opfinde algoritmer uafhængigt af det konkrete programmeringssprog, og samtidig give dem den fornødne viden til at oversætte deres algoritmer til et program der kan køres. Ideen er at tage den proces en erfaren programmør genenmgår mentalt, og stilladsere den for eleverne ved at give dem fysiske objekter at flytte rundt på.

1.1 Teoretisk baggrund

Inden jeg præsenterer min problemformulering, vil jeg kort gennemgå de mest centrale teorier, jeg har tænkt mig at benytte.

1.1.1 Unplugged Computing

Det er svært at undervise i programmering[12]. Over årene er der mange der har forsøgt at løse den udfordring der ligger i ikke blot at oplære eleverne i de forskellige sprogelementer og deres funktion, men også at undervise i hvordan disse elementer bedst anvendes, og hvordan man *tænker som en programmør*. Et af de mange svar der er blevet udviklet er en metode der hedder *unplugged computing*, og som groft sagt består i at tage computeren ud af undervisningen, og i stedet bruge spil og lege til enten at simulere hvad der sker i computeren, eller øve bestemte måder at tænke på[4]. Disse aktiviteter kan involvere fysisk aktivitet, eller de kan foregå relativt stillesiddende – de fælles elementer er fraværet af computeren og brugen af leg og spil.

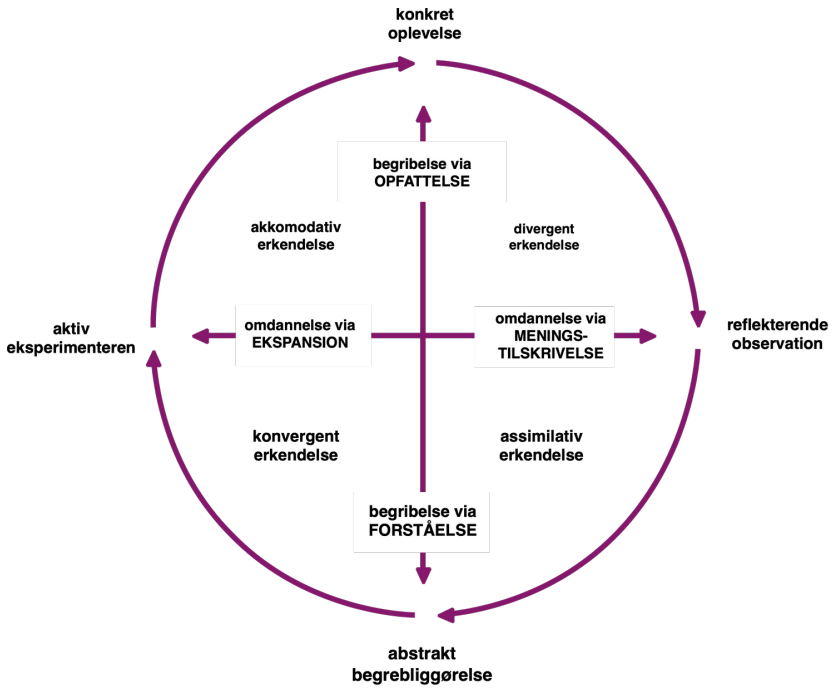
Forsøg med unplugged computing har vist øget engagement fra eleverne[6]. Busuttil og Formosa bemærker bl.a. at eleverne er meget engagerede når aktiviteterne involverer *hard fun*, altså en sjov udfordring som kræver at de tænker sig om[6].

1.1.2 Erfaringslæring

John Dewey mente at man lærer ved at gøre os erfaringer og reflektere over dem – man agerer, man gør et eller andet ved verden, og så opdager man hvad konsekvenserne af ens handling er[2]. Dewey mener at det er en lærers opgave at sætte rammer for at eleverne kan eksperimentere og erfare, men også reflektere over erfaringerne og forbinde dem med teori[2]. Han opstiller læring som en proces bestående af fire faster: Først befinder man sig et stadie af uligevægt, hvor man undrer sig over noget. Dernæst undersøger og observerer man for at finde ud af noget om det man undrer sig over, og i tredje fase drager man så nogle konklusioner om fænomenet. Fjerde fase består i at gå ud og eksperimentere videre baseret på disse konklusioner[2].

David Kolb bygger ovenpå bl.a. Deweys teori, men låner også begreber og koncepter fra Piaget og Lewins læringsmodeller[9]. Han opstiller også læring som en proces med fire stadier, men hans er lidt anderledes end Deweys. Hos Kolb starter læring med en konkret oplevelse, efterfulgt af reflekterende observation. Observationerne assimileres og 'sættes på begreb', de bliver til abstrakte koncepter, og endelig eksperimenteres der så ud fra disse koncepter[9]. De fire stadier kan ses på figur 1.

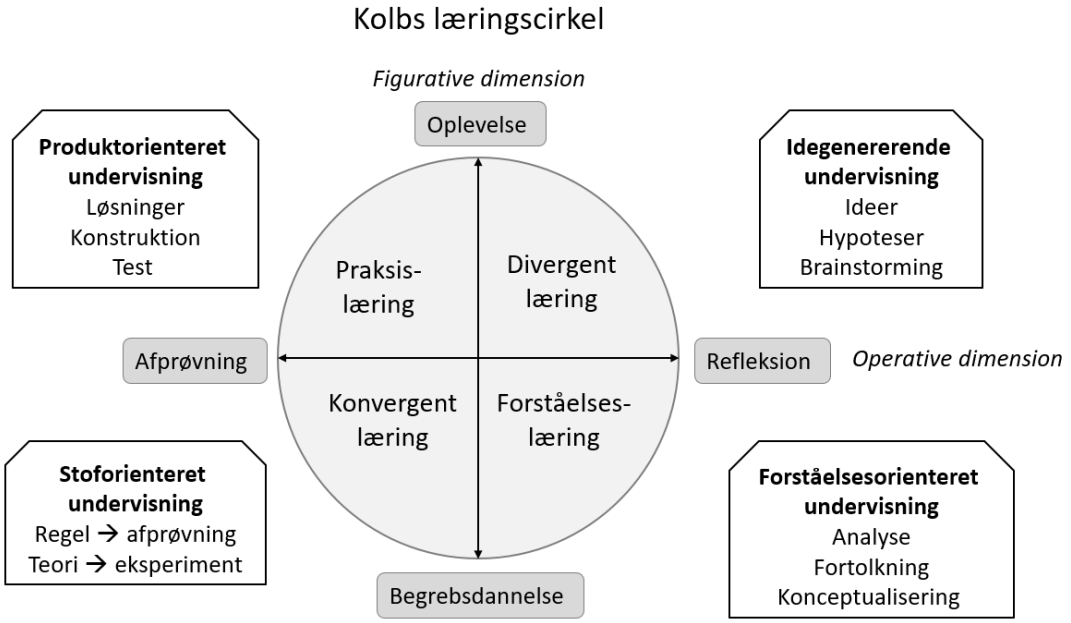
Kolb uddyber og systematiserer også i højere grad hvad der sker i og mellem stadierne. På figur 1 ser vi to akser spændt ud mellem de fire stadier. Den lodrette akse, mellem oplevelse og begrebsliggørelse, adskiller to måder at begribe verden på – ved at opleve den direkte, eller via begreber og koncepter. Den vandrette akse, mellem refleksion og eksperimenteren, adskiller to måder at omforme vores forståelse af verden på. Refleksion foregår internt, mens eksperimenteren handler om at påvirke den ydre verden[9].



Figur 1: Kolb's læringscirkel

Endelig beskriver Kolb fire former for erkendelse, som hver findes i spillerummet mellem to stadier. *Divergent erkendelse* forekommer når man begriber en oplevelse gennem opfattelse, og omformer den gennem refleksion og meningstilskrivelse. *Assimilativ erkendelse* opstår når man begriber gennem abstrakte koncepter, og omformer dem gennem meningstilskrivelse. *Konvergent erkendelse* opstår når abstrakte koncepter udvides gennem aktiv eksperimenteren, og *akkomodativ erkendelse* forekommer når en oplevelse begribes gennem opfattelse, og omdannes gennem udvidelse[9].

Beck og Ebbensgaard har lavet en bearbejdet version af Kolb's læringscirkel, som kan ses i figur 2. De mener at det at begribe gennem abstrakte koncepter og omforme gennem meningstilskrivelse godt kan føre til akkomodativ erkendelse snarere end assimilativ, hvis den lærende gennem processen får en helt ny forståelse af et begreb. Tilsvarende mener de også at der i feltet mellem aktiv eksperimenteren og konkret oplevelse kan forekomme assimilativ læring, hvis et eksperiment f.eks. bruges til at vælge mellem kendte tilgange[2].



Figur 2: Kolb's læringscirkel, bearbejdet af Beck & Ebbensgaard

1.1.3 Unplugged Computing som erfaringslæring

Jeg mener at der er en klar sammenhæng mellem unplugged computing og Kolbs erfaringslæring. I unplugged computing er undervisningen meget induktiv[6] – eleverne arbejder i grupper, og skal hovedsageligt selv finde frem til løsninger eller viden ved at eksperimentere, eller ved at producere noget[6]. Det er et vigtigt princip i unplugged computing, at eleverne oplever at de selv kan finde frem til løsninger på de problemer de bliver stillet over for[11]. Eksperimenter og erfaring er altså centrale elementer i unplugged computing.

1.2 Problemformulering

I mit projekt vil jeg tage inspiration fra unplugged computing, og forsøge at designe aktiviteter der kan hjælpe eleverne med at bruge de kontrolstrukturer de kender til at løse problemer de ikke har set før. Problemformuleringen lyder som følger:

> *Undersøge om læringsspil inspireret af Unplugged Computing kan hjælpe eleverne i Programmering B til at gå fra at læse og forstå kode, til at skrive det.*

2 Metode

For at besvare min problemformulering, har jeg udført en aktion[1] i min programmeringsklasse, 2.i. Selve forløbet handlede om algoritmer der arbejder med lister, og indeholdt to forskellige aktiviteter inspireret af Unplugged Computing. Til sidst i forløbet skulle eleverne også implementere de algoritmer de havde arbejdet med i programmeringssproget Python.

Da jeg efterfølgende skulle analysere forløbet, var der særligt to læringsteorier der stod frem som værende interessante. Det drejede sig om Dewey og Kolbs teori om erfaringslæring[2, 1], og om Lave og Wengers teori om praksislæring og praksisfællesskaber[1]. Valget af teori påvirker hvilke aspekter af undervisningen man kan undersøge: Med Lave og Wenger kan jeg tale om de dynamikker som opstod i grupperne, og hvordan eleverne brugte hinanden for at lære. Med Dewey og Kolb kommer det i stedet til at handle om de undringer jeg har forsøgt at fremprovokere, og om hvordan forløbet har givet eleverne mulighed for at eksperimentere, erfare, og reflektere.

Jeg endte med at sætte mig fast på Kolbs erfaringslæring som den teori jeg primært har brugt, og kommer derfor også til at bruge nogle af Deweys begreber. Praksisfællesskaberne er interessante, men forløbet handlede om at eleverne skulle lære ved at afprøve og eksperimentere, altså gennem erfaring. Derfor mener jeg at Kolbs teori om erfaringslæring bedre kan hjælpe mig til at undersøge om forløbet har fungeret som ønsket.

I bilag A findes en detaljeret FIMME-plan for forløbet, men jeg vil også her beskrive de nævnte aktiviteter, samt nogle af de didaktiske overvejelser jeg har gjort mig i forbindelse med opbygningen af undervisningstimerne.

2.1 Design af aktiviteter

Forløbet består som sagt af to aktiviteter inspireret af Unplugged Computing. Hver af disse aktiviteter ender ud i at eleverne producerer en beskrivelse af en algoritme. Til sidst i forløbet skal eleverne så programmere de algoritmer der er blevet arbejdet med. Den første aktivitet fører til en relativt uformel, pseudokode-agtig beskrivelse af hver algoritme. Den anden aktivitet fører til et flowdiagram, som er en mere formel og præcis beskrivelse. Man kan sige at programmeringsøvelsen til sidst er et tredje trin på vejen fra uformel til formel beskrivelse – til sidst får eleverne beskrevet deres algoritmer så præcist, at en computer kan udføre dem.

2.1.1 Aktivitet 1: Rollespil med kort

Den første aktivitet går ud på at bruge spillekort til at repræsentere værdier i et program – i dette tilfælde udelukkende heltal – og at rollespille hvad der sker når en computer eksekverer et program. To elever agerer til sammen computeren: De har hver især lov til at samle et kort op fra bordet, sammenligne det med den anden elevs kort, og lægge kortet ned igen. Resten af gruppen spiller programmøren, som skal give instruktioner til computeren. Der er klare grænser for hvilke informationer der må udveksles mellem deltagerne – computeren må fortælle programmørne hvilken af de to elever der har det største kort, og programmørerne må fortælle computeren hvilke

kort der skal samles op af hvem, hvornår de skal sammenlignes, og hvor og hvornår kortene skal lægges tilbage på bordet.

For at eleverne kan lære reglerne at kende, udføres aktiviteten først fælles i klassen med to frivillige elever som computer, mens hele resten af klassen programmerer. Opgaven er at finde det største tal blandt en række af kort, som ligger med talsiden nedad.

Herefter fordeles eleverne i grupper, som hver får en bunke kort. Hver gruppe får tildelt en opgave, som skal løses indenfor samme regler. Opgaverne lyder:

- Find det næststørste tal i en liste
- Find to tal i listen, der summer til 12
- Fordel tallene i listen, så alle de lige tal ligger til venstre for alle de ulige tal
- Fordel tallene i listen, så alle tal under 7 ligger til venstre for alle tal over 7

Eleverne får desuden besked på at nedskrive instruktioner til deres løsning, så de andre grupper kan afprøve dem og se om løsningen også virker for dem. Eleverne har ikke arbejdet med pseudokode før, så der bliver ikke lagt vægt på at de skal skrive pseudokode, blot at de kan bruge en kode-lignende struktur, og at de gerne f.eks. må skrive noget der ligner en if-sætning eller en løkke i deres instruktioner.

Formålet med at nedskrive instruktionerne og afprøve hinandens algoritmer, er at opdage noget om vigtigheden af præcise forklaringer, som så skal føre hen til mere formelle måder at nedskrive algoritmer på.

2.1.2 Aktivitet 2: Spilleplader og flowdiagrammer

Den anden aktivitet handler om at lave en mere formaliseret version af algoritmerne der blev udviklet i den første aktivitet, i form af en spilleplade og et flowdiagram for hver algoritme. I den første aktivitet var det eleverne der agerede variable – nu får hver variabel et navn, og bliver til et felt på en spilleplade, hvor der også er pladser til listen af tal. Tidligere har eleverne implicit holdt styr på hvor de er nået til i listen, men nu bliver tællevariable også navngivet og skrevet ned på spillepladen, og det skal gøres eksplicit hvornår de skal inkrementeres.

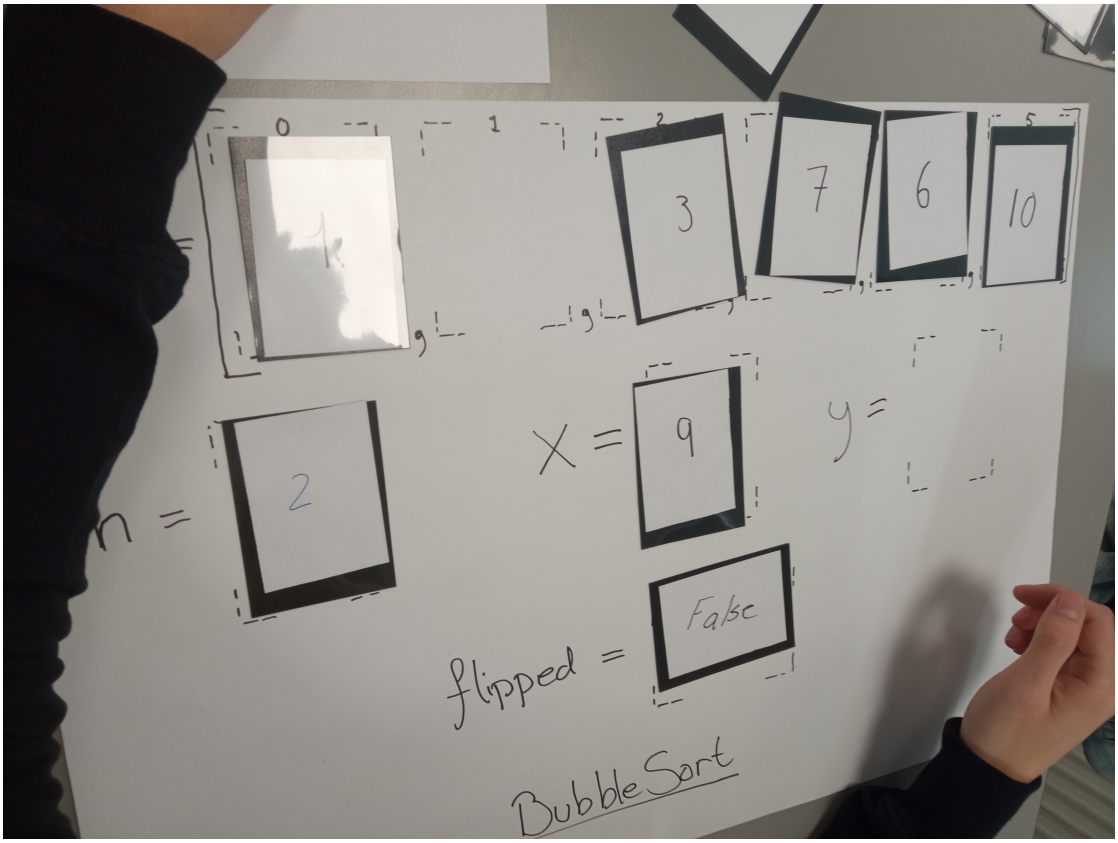
I denne aktivitet ligger kortene ikke længere med talsiden nedad. Nu er alt synligt, og det handler om at gøre instruktionerne så præcise at de kan udføres, uden at man nødvendigvis ved hvad målet er.

Igen skal eleverne introduceres til aktiviteten og dens spilleregler. Til det formål får de udleveret to allerede færdiggjorte algoritmer med spilleplade og flowdiagram. Figur 3 viser et eksempel på sådan en spilleplade, for algoritmen BubbleSort, og figur 4 viser det tilhørende flowdiagram. Eleverne får mulighed for at afprøve begge de udleverede algoritmer, BubbleSort og InsertionSort, før de går igang med deres egne algoritmer i grupperne.

Denne gang er produktkravet for hver gruppe en spilleplade og et flowdiagram. Efterfølgende er der afsat tid til at eleverne kan afprøve de andre gruppers algoritmer.

2.2 Forløbets opbygning

Da jeg planlagde forløbet, vidste jeg på forhånd at det skulle foregå i en uge i marts, hvor jeg havde ni skemalagte timer med klassen fordelt på tre dage. De tre timer lange undervisningsgange gav god tid til både at lære reglerne for de to aktiviteter, og til at udføre eksperimenter og udarbejde produkter. En FIMME-plan findes i bilag A. I dette afsnit vil jeg gennemgå forløbets opbygning lidt mere overordnet, og forklare hvorfor jeg har bygget det op på den måde.



Figur 3: En elev afprøver BubbleSort-algoritmen på et spillebræt

Hver undervisningsgang er arrangeret sådan at der først er noget tavleundervisning hvor jeg introducerer hvad vi skal lave, og gennemgår eller repeterer nødvendig viden. Derefter er der nogle aktiviteter eller opgaver, og til sidst er der en opsamling hvor vi diskuterer hvad eleverne har erfaret under arbejdet. Opsamlingen til sidst er vigtig – den skal sikre at eleverne også får reflekteret over det de har fundet ud af, hvilket er et vigtigt trin i erfaringslæring[2, 9].

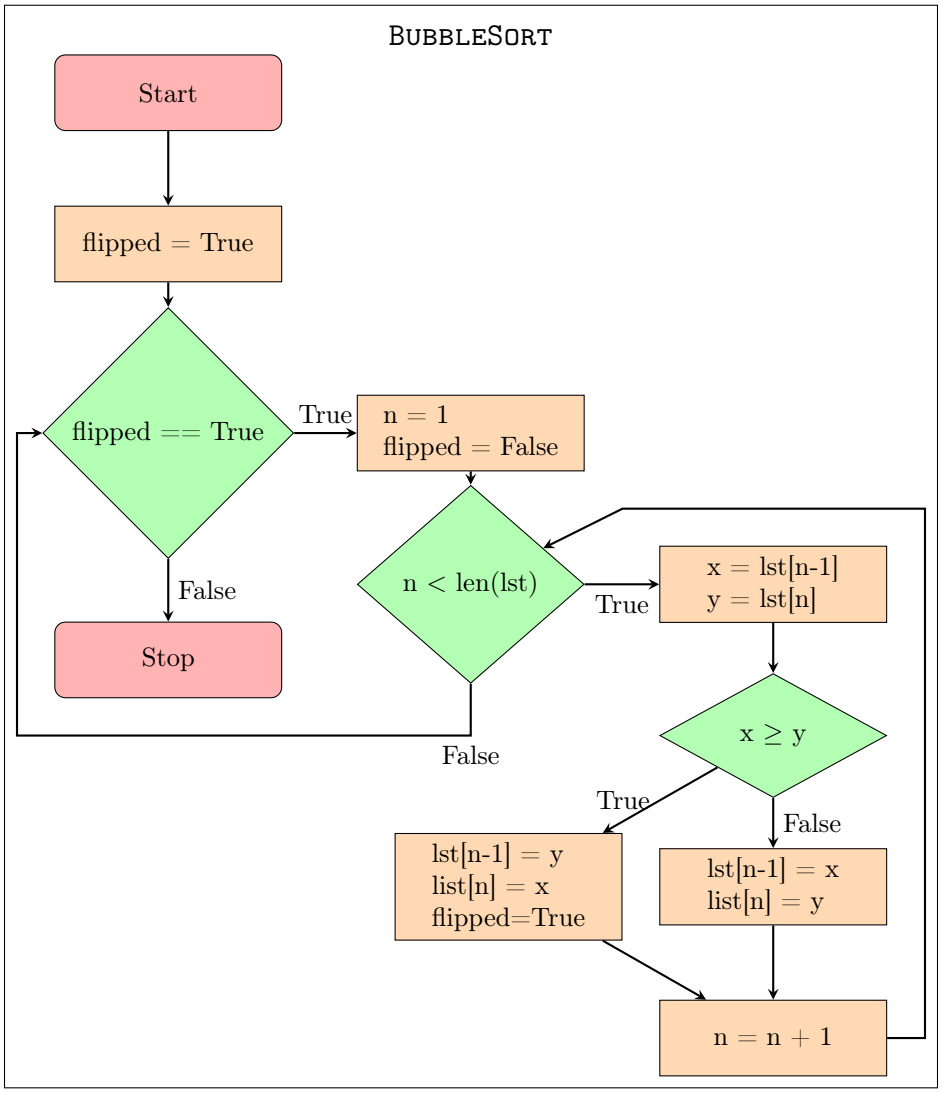
Til første undervisningsgang planlagde jeg en introduktion til emnet, der skulle veksle mellem IRE- og IRF-undervisning[1]. IRF-undervisningen skulle bringe elevernes forforståelse på banen, hvilket falder fint sammen med Dewey's ideer om at inddrage elevernes erfaringer fra deres hverdagsliv som udgangspunkt for erfaringslæring[2]. IRE-strukturen brugte jeg primært til at repetere viden fra tidligere undervisningsforløb.

Klassediskussionen skulle ende ud i en definition af begrebet *algoritme*, illustreret med et eksempel fra Teach London Computing, hvor man ved at følge en simpel instruktion kan få en robot til at forsvinde fra et billede[8]. Eksemplet gennemgås sammen med eleverne – vi tæller robotter før og efter algoritmen er kørt, og diskuterer på IRF-form hvordan tryllekunsten virker. Den vigtige pointe er at tricket også virkede, selvom vi ikke forstod hvordan – en computer forstår heller ikke hvorfor den gør hvad den gør, men derfor skal algoritmerne stadig virke.

Resten af første undervisningsgang går med kortspils-aktiviteten beskrevet i afsnit 2.1.1, efterfulgt af en kort opsamling, hvor jeg på IRE-form spørger ind til hvilke erfaringer de har gjort sig i forhold til at følge andre gruppers instruktioner.

BubbleSort

Input: lst (liste af tal)
Efter endt kørsel vil lst være sorteret



Figur 4: Udleveret flow-diagram for algoritmen BubbleSort

Anden undervisningsgang starter med en kort genopfriskning af første undervisningsgang, og derefter repetition af flowdiagrammer – klassen har arbejdet med dem før, men der er gået et stykke tid. Dette er igen klassisk IRE-undervisning. Derefter laver vi spilleplade-aktiviteten beskrevet i afsnit 2.1.2. Da eleverne skal afprøve de udleverede sorteringsalgoritmer, bliver de også bedt om at tælle hvor mange gange de sammenligner to tal. Resultatet af disse optællinger bliver brugt til en lærerstyret klassediskussion om køretidskompleksitet. Derefter fortsætter eleverne med aktiviteten. Undervisningen slutter endnu en gang med en opsamling, hvor jeg spørger ind til deres oplevelser med at følge hinandens flowdiagrammer.

Tredje og sidste undervisningsgang starter ud med tavleundervisning om hvordan man oversætter fra flowdiagram til kode. Tanken er at det skal være IRE-undervisning med udgangspunkt i et eksempel – jeg tegner et flowdiagram på tavlen, og programmerer det tilsvarende program med input fra eleverne. Vi fortsætter med at snakke om hvordan vi så ved om vi har gjort det rigtigt, og skriver nogle tests af programmet. Denne form for demonstration er inspireret af ideen om *worked examples*, hvor eleverne får mulighed for at se hele programmeringsprocessen, inklusive de fejl der måtte blive lavet og rettet undervejs[7, 5].

Det næste segment er individuel opgaveløsning, hvor eleverne prøver at implementere og teste deres egne algoritmer, mens jeg bevæger mig rundt i lokalet og hjælper. Timen slutter med en opsamling på hele forløbet, hvor vi sammenligner de tre forskellige måder vi har repræsenteret algoritmer på. Her er pointen at gøre det meget tydeligt, at en algoritme *ikke* er det samme som et program eller et flowdiagram, men eksisterer uafhængigt af repræsentationsformen.

2.3 Gruppedannelse og differentiering

I dette forløb ville jeg gerne have at eleverne sammen diskuterede og legede sig frem til løsninger. Jeg valgte at inddele eleverne så hver gruppe var nogenlunde homogen i forhold til fagligt niveau, altså hierarkisk differentiering[3]. Formålet med dette var at undgå, at en eller to fagligt stærke elever løste det hele, mens resten kun deltog passivt. Der måtte gerne være nogen forskel i fagligt niveau, f.eks. nogle virkelig dygtige elever sammensat med nogle middel-dygtige elever, men spændet skulle ikke være for bredt.

Derudover ville jeg gerne have at grupperne var lidt større end de sædvanlige 2-3 personers grupper vi har brugt i faget indtil nu. En del af forløbet indvolverede at to eller tre elever skulle rollespille bestemte funktioner, og her ville jeg gerne sikre at der så også var to eller tre elever "til overs" til at dirigere arbejdet. Med en klasse på 19 elever var det oplagt at inddele dem i fire grupper af størrelse ca. 5.

Det meste af undervisningstiden gik med at arbejde med fire forskellige problemer, beskrevet i afsnit 2.1.1, som eleverne skulle løse ved at finde på algoritmer. De fire problemer gav en oplagt mulighed for at differentiere undervisningen – nogle var simplere end andre. To af problemerne handlede om at finde enkelte elementer i en liste, der opfyldte et kriterie (forholdsvis ligetil), mens de andre to handlede om at partitionere en liste baseret på et kriterie (lidt mere komplekst). Partitioneringsopgaverne gik altså til de to grupper med meget fagligt dygtige elever, mens de andre to opgaver gik til grupperne med knap så dygtige elever.

2.4 Indsamling af empiri

Under forløbet har jeg indsamlet data på tre måder: Observation, elevernes produktioner, og et kort interview med udvalgte elever. Elevernes læringsudbytte har jeg primært undersøgt baseret på deres produktioner og mine observationer under forløbet, mens interviewet efterfølgende har fortalt mig noget om elevernes engagement.

Under selve forløbet har jeg det meste af tiden fungeret som vejleder, mens eleverne har arbejdet i grupper. Dette har givet rig mulighed for at træde tilbage og observere eleverne, men det er selvfølgelig altid sådan, at en underviser ikke kan se alt, og ikke kan være en helt neutral observatør. Når jeg er henne ved en gruppe og vejlede, så kan jeg for det første ikke have øjne på de andre grupper, og for det andet er jeg heller ikke en udenforstående observatør over for den gruppe jeg er ved – min tilstedeværelse påvirker situationen.

Under forløbet har eleverne produceret en række artefakter: Pseudokode, flowdiagrammer, spilleplader, og til sidst programkode. Med undtagelse af programkoden er disse produktioner udarbejdet i grupper, og fortæller derfor ikke så meget om den enkelte elevs læringsudbytte, men giver et overordnet billede af gruppens ståsted. Eksempler på elevproduktioner kan ses i bilag B og C.

Endelig har jeg, efter forløbets afslutning, interviewet fem elever. Interviewet var forholdsvis ustruktureret, da jeg primært var interesseret i at finde ud af hvad der fremstod som vigtigt for eleverne ved forløbet. Hovedemnet jeg spurgte ind til var motivation og engagement, og om emnet føltes tilgængeligt for dem.

3 Klassen

2.i er en klasse bestående af 19 elever på studieretningen Kommunikation/IT A og Programmering B. De er en meget engageret klasse, som generelt laver deres lektier og arbejder koncentreret i timerne. Ca. halvdelen af dem deltager uopfordret i klassediskussioner og stiller interessespørgsmål, mens resten skal opfordres lidt før de deltager i det store klassefællesskab.

Eleverne giver generelt udtryk for at de godt kan lide programmering, og at de synes det er et vigtigt fag, men også at det er svært.

3.1 Grupperne

Eleverne arbejdede under forløbet i fire store grupper (en gruppe på 4 elever, og tre grupper på 5 elever). Grupperne var baseret på deres siddepladser, da dette passede godt med den fordeling jeg ønskede – de sidder omkring fire rektangulære "øer" af borde, hvor de er nogenlunde fordelt efter fagligt niveau.

- **Gruppe 1** bestod af fem engagerede elever – tre af de dygtigste programmører i klassen, og to som er på et middelniveau fagligt, og også ret aktive.
- **Gruppe 2** bestod af fem elever som deltager i timerne, men ikke nær så aktivt som gruppe 1. To af dem er meget fagligt stærke, mens de andre tre ligger på et middel-niveau.
- **Gruppe 3** bestod af fem elever, som alle ligger på lavt til middel fagligt niveau, og som bedst kan beskrives som fagligt usikre – i de fleste tilfælde kan de egentlig godt løse opgaverne, men de tror ikke selv på det, og søger meget bekræftelse fra mig.
- **Gruppe 4** bestod af de fire mest stille elever i klassen. To af dem er fagligt meget svage, mens de andre to er på et middel-niveau. Den ene af de to fagligt svage elever har det med at blive distraheret og lave noget andet, mens den anden arbejder koncentreret med opgaveløsning, men nemt går i stå og ikke kan lide at spørge om hjælp.

4 Analyse af forløbet

I dette afsnit vil jeg analysere forløbet med udgangspunkt i David Kolbs teori om erfaringslæring. Jeg vil bruge Kolbs læringscirkel (figur 1), og i nogle tilfælde Beck og Ebbengaards lettere bearbejdet version af samme[2] (figur 2), til at analysere mindre segmenter af undervisningen, og vil så afslutningsvis forsøge at drage nogle konklusioner om forløbet som helhed.

Kolb definerer overordnet læring som *'den proces, hvorved erfaring omdannes til erkendelse'*[9]. Kolb indeler denne proces i fire stadier spændt ud på en cirkel – konkret oplevelse, reflekterende observation, abstrakt begrebliggørelse, og aktiv eksperimenteren – og mellem hvert nabopar af stadier beskriver han så fire typer erkendelse. [9]. Disse kan ses på figur 1.

Mit forløb benytter sig af alle fire typer erkendelse, men ikke i samme grad. Meget af tiden går med at eleverne eksperimenterer og oplever resultaterne, altså det Kolb ville kalde akkomodativ erkendelse, og som Beck og Ebbensgaard i deres bearbejdede version vil kalde praksislæring (figur 2).

4.1 Klassediskussion om algoritmer

Det første segment af forløbet var en lærerstyret klassediskussion, beskrevet i afsnit 2.2. Her kunne jeg observere aktiv deltagelse fra alle hjørner af klasselokalet. Denne diskussion var i høj grad baseret på erfaringer eleverne har gjort sig, i og uden for klasselokalet, og kan med Kolbs ord beskrives som divergent læring – eleverne havde oplevet noget, og nu forsøgte vi i fællesskab at sætte nogle ord på det og reflektere over hvad det var for noget. Robot-eksemplet førte til en del latter og diskussion af hvad der helt præcist talte som en robot, også fra nogle af de elever der ellers ikke siger så meget. Det er et eksempel som ikke kræver nogen faglig viden, hvilket nok har hjulpet med at gøre det tilgængeligt for de elever der ikke er så fagligt selvsikre.

Herefter gik vi videre til at opstille en lidt mere formel definition af begrebet *algoritme*. Her er tale om assimilativ læring, hvor vi forsøger at tage alle vores forskellige observationer og refleksioner, og få dem til at passe ind under et mere generelt begreb.

4.2 Kortspils-aktiviteten

Med vores abstrakte algoritmebegreb i hånden kunne vi nu gå videre til kortspils-aktiviteten beskrevet i afsnit 2.1.1. Eleverne skulle udføre et eksperiment og gøre sig nogle nye observationer.

Introduktionen til spillet foregik med hele klassen samlet om en af bord-øerne, så alle kunne se. Her var en del forvirring om reglerne, som måtte opklares undervejs – et par af eleverne kan godt lide at tage instruktioner meget bogstaveligt, og finde smuthuller der kan spare dem noget arbejde. Det er en fin egenskab for en programmør, men ikke så hjælpsomt i undervisningssammenhæng. For eksempel var kortene med tal lige præcis gennemsigtige nok til at man kunne se tallene når de blev holdt op mod lyset fra vinduet, hvilket gjorde det meget nemt at finde det største tal uden egentlig at engagere sig med spillets regler.

Hver af de fire grupper havde fået deres egen opgave, beskrevet i afsnit 2.1.1. Grupperne arbejdede selvstændigt, mens jeg mere trådte ind i en vejleder-rolle – jeg bevæger jeg rundt mellem grupperne, og hjælper når nogen ser ud til at være gået i stå. På Beck's læringscirkel bevæger vi os mellem opgaverummet og dialogrummet[1] – jeg brugte det meste af tiden hos den ene eller den anden gruppe, hvor jeg enten stillede udfordrende spørgsmål der lagde op til diskussion i gruppen, eller gav hints til hvordan de kunne overkomme bestemte udfordringer. Eleverne arbejdede udforskende og eksperimenterende. De havde et mål de skulle nå, men de havde ikke travlt, og stemningen var afslappet og munter.

I forhold til Kolbs læringscirkel, så bevæger vi os rundt på den øverste halvdel: Eleverne er i en cyklus af eksperimenteren, opfattelse, og meningstilskrivelse. Gruppen har en ide, de afprøver

den, og ser om den virker. Hvis ikke, så prøver de at finde ud af hvorfor, hvilket fører til nye ideer. Der er som sådan ikke nogen abstrakt begrebsliggørelse her, men det er også fint – ideen er at de skal udføre mange små eksperimenter og reflektere over dem, og senere kan vi så i fællesskab forsøge at sætte det i system og begrebsliggøre det.

4.2.1 Nedskrivning af algoritmer

Efterhånden som hver gruppe nåede frem til en fungerende algoritme, skulle de også skrive instruktionerne ned. Det førte til en masse refleksion og abstrakt begrebsliggørelse, da de nu skulle oversætte hverdagssprog til en lidt mere struktureret tekst. De skulle f.eks. opdage, at når de i hverdagssprog ville sige 'og så bliver man bare ved med at gøre sådan indtil man er igennem alle tallene', så kan det oversættes til en løkke med en bestemt betingelse. Her er der altså tale om forståelseslæring eller assimilativ erkendelse – det handler om at genkende allerede kendte koncepter i en ny kontekst, altså at tage nye observationer og finde ud af hvordan de hænger sammen med abstrakte begreber.

Jeg måtte hjælpe en del med formuleringen af instruktionerne, da klassen ikke har arbejdet med pseudokode før, og ikke havde fået en specielt grundig gennemgang af dette. Tvivlen handlede primært om hvor meget instruktionerne skulle brydes ned i detaljer – kan man f.eks. godt skrive 'byt om på de to kort', eller skal man udpensle hvordan det foregår? Her henviste jeg til, at det primære formål med disse instruktioner var at klassekammerater fra de andre grupper skulle kunne følge dem, og at det derfor f.eks. var fint at skrive ting som 'byt om på de to kort'.

4.2.2 Afprøvning af andres algoritmer

Timen bød på ét eksperiment mere. Eleverne skulle ud og afprøve de algoritmer som andre grupper havde skrevet ned, og finde ud af om deres instruktioner kunne bruges. Her opstod for alvor den *uligevægt* som Kolbs forgænger Dewey mener er forudsætningen for al læring[2]. Det handler ikke så meget om at opdage at de andres instruktioner er svære at følge, men mere om at en anden gruppe kommer og siger 'jeres instruktioner giver ikke mening' – den gruppe der har udarbejdet instruktionerne, forstår godt hvad de selv har skrevet, og bliver overraskede og lidt utilfredse med at få den form for kritik. Reaktionen var i alle tilfælde at gribe blyant og hviskelæder, og lave rettelser til instruktionerne. Her kan man finde en hel lille omgang rundt i Kolbs læringscirkel – eleverne har opdaget at deres klassekammerater ikke kan forstå deres instruktioner (konkret oplevelse), så de spørger nærmere ind til hvad der er tvetydigt (observerende refleksion). Derefter prøver de at komme på mere præcise måder at skrive det på (abstrakt begrebsliggørelse), og udleverer så de reviderede instruktioner til yderligere afprøvning (aktiv eksperimenteren).

Ved den efterfølgende opsamling, gav eleverne udtryk for at det var meget svært at følge andres instruktioner – der var mange tvetydigheder og manglende informationer. Dette lagde fint op til næste dags arbejde, som netop handlede om at tage de udarbejdede algoritmer, og beskrive dem på en mere formel og præcis måde. En elev sagde desuden til opsamlingen at *'det var meget tid at bruge på bare at lege med kort'* – måske forbindelsen til selve programmeringen ikke var helt tydelig for eleven på dette tidspunkt.

Ved det efterfølgende interview mente en elev at *'kortene gjorde det nemmere at forstå hvad det handlede om'*. I hvert fald denne elev har altså godt kunnet se sammenhængen til sidst i forløbet, men det kan være det skal gøres endnu tydeligere fra starten.

4.3 Spilleplade-aktiviteten

Den anden unplugged aktivitet, beskrevet i afsnit 2.1.2, byggede videre på eksperimentet fra afsnit 4.2.2. Eleverne havde erfaret at det er vanskeligt at skrive instruktioner ned, der er præcise nok til at andre kan følge dem. Nu opstillede vi så en mere formel og præcis måde at beskrive algoritmer på, altså i form af flowdiagrammer (abstrakt begrebsdannelse), og afprøvede denne (aktiv eksperimenteren). Overordnet set førte dette segment til konvergent erkendelse: Oplevelsen forstås ved hjælp af et abstrakt symbolsprog, og omformes gennem eksperimenteren.

Inden eleverne for alvor gik i gang, var der et kort segment med tavleundervisning, hvor vi repeterede det eleverne tidligere havde lært om flowdiagrammer. Vi gennemgik også i fællesskab hvordan man læser og udfører en algoritme på dette format.

4.3.1 Afprøvning af sorteringsalgoritmer

Hver gruppe fik udleveret en spilleplade og flowdiagram for en sorteringsalgoritme, som de skulle køre igennem, og undervejs tælle antallet af sammenligninger. Vi startede altså med aktiv eksperimenteren og praksislæring, hvorefter vi bevægede os over i divergent læring, da eleverne skulle diskutere – først i grupper og så med resten af klassen – hvad man kan sige om algoritmerne baseret på deres optælling af sammenligninger. Segmentet blev afrundet med en gennemgang af begrebet *køretidskompleksitet* ved tavlen, så der nåede vi til abstrakt begrebsliggørelse, og kom dermed rundt om det meste af cirklen. Det næste eksperiment handlede dog ikke om køretidskompleksitet, så man kan ikke sige at der skete konvergent læring om netop dette emne.

4.3.2 Egne spilleplader og flowdiagrammer

Eleverne skulle nu udarbejde deres egne spilleplader og flowdiagrammer baseret på arbejdet fra kortspils-aktiviteten. Her startede de fleste grupper med at genopfriske hvad de lavede sidst, hvilket er en form for reflekterende observation. Mange elever kunne ikke huske hvordan deres algoritme virkede. Nogle forsøgte at gå i gang med den nye opgave uden rigtig at vide hvad de lavede, og opdagede hurtigt at det de lavede var noget rod. Andre kørte algoritmen igennem baseret på gårsdagens nedskrevne instruktioner, og blev undervejs endnu en gang konfronteret med de tvetydigheder der nu engang måtte være.

Den proces der foregik i dette segment mindede meget om hvad jeg har beskrevet i afsnit 4.2.2. Eleverne skrev noget ned, afprøvede det, opdagede at det ikke helt virkede, og gik tilbage til tegnebrættet. Det er den samme cyklus fra eksperiment til oplevelse til observerende refleksioner, og videre til et nyt eksperiment. I starten var det primært gruppen selv der afprøvede deres arbejde, og senere hen blev andre grupper involveret – her tog fejlfindingsprocessen for alvor fart. Det der skete her er *iterativ udvikling*, og det opstod helt af sig selv og uden min indblanding eller intension.

Jeg lagde mærke til at nogle af grupperne blev så optagede af opgaverne, at de kom ud i nogle decideret højlydte diskussioner af hvordan løsningerne skulle se ud. Den form for livlig faglig debat tyder på at eleverne er virkelig motiverede – de deltager aktivt i et praksisfællesskab[10].

4.3.3 Opsamling på de to aktiviteter

Som afslutning på andendagen tog vi en snak i klassen om de to måder at nedskrive algoritmer på. Her fik vi igen sat nogle fagbegreber på hvad det var eleverne havde produceret, og hvad der gik galt – vi sluttede altså af med forståelseslæring.

4.4 Implementation af algoritmer

Som den sidste fase af forløbet, skulle eleverne nu anvende deres viden om de konkrete algoritmer de havde arbejdet med, og altså prøve at implementere dem i programmeringssproget Python. Det var dette segment, hvor jeg skulle se om jeg havde opnået det mål jeg satte mig for i begyndelsen – altså om eleverne ville være i stand til at udtrykke deres løsninger ved hjælp af de tilgængelige kontrolstrukturer i et programmeringssprog.

Kolb er ikke specielt interesseret i at undersøge resultaterne af undervisning på en behavioristisk måde. Han mener at læring er en proces, der hele tiden skabes og genskabes[9]. Hvis jeg skal bedømme elevernes udbytte af min undervisning fra dette perspektiv, så er jeg interesseret i om de er i stand til at eksperimentere og erfare videre, altså om de kan blive ved med at bygge oven på det de har arbejdet med i forløbet.

Under tidligere forløb har eleverne typisk stillet meget brede spørgsmål der tyder på manglende overblik over hvor de er på vej hen. Under arbejdet med at implementere algoritmerne i dette forløb, kom eleverne længere på egen hånd, og havde langt mere konkrete og målrettede spørgsmål. I stedet for at spørge til 'hvad skal jeg nu', eller 'hvorfor virker det ikke', handlede spørgsmålene om bestemte dele af koden eller flowdiagrammerne. Dette mener jeg netop viser, at de eksperimenterer videre ud fra ny abstrakt viden, som de har tilegnet sig gennem erfaring.

Eleverne brugte generelt deres flowdiagrammer som vejledning til implementationen, men mange steder fandt de ud af at flowdiagrammerne ikke var egnede til direkte oversættelse til løkker og if-sætninger. Her var altså endnu et øjeblik med uligevægt, hvor eleverne måtte reflektere over deres oplevelse, og så se nærmere på de generelle regler for sammenhængen mellem flowdiagram og kode, hvilket igen er en form for abstrakt begrebsliggørelse – de skulle finde ud af hvordan de konkrete problemer de oplevede, passede ind i de mere abstrakte regler.

Senere i processen afprøvede eleverne deres programmer, og her skete der endnu engang samme fejlfindingsproces som er beskrevet i afsnit 4.2.2 og 4.3.2. Nogle fejl skyldtes at de ikke havde fulgt flowdiagrammerne helt tæt, mens andre skyldtes fejl i selve flowdiagrammet som ikke var blevet opdaget når mennesker har skulle udføre det – f.eks. off-by-one-fejl, som ofte bliver overset fordi mennesker har det med at kigge knap så nøje på flowdiagrammet når først det generelle mønster er lært, og målet er i syne. Her var endnu en mulighed for reflekterende observation, omend kun for de af eleverne der faktisk oplevede denne type fejl.

Alle elever nåede at implementere mindst to algoritmer (en af deres egne, og en af de udleverede sorteringsalgoritmer). En elev har i interviewet sagt at *'det var nemmere at programmere det efter vi havde prøvet det af på så mange forskellige måder'*, hvilket passer meget godt med hvad jeg selv observerede i undervisningen. Særligt en elev, som ofte går i stå når han skal programmere, var i stand til at implementere algoritme nummer to mere eller mindre på egen hånd, da jeg først havde gennemgået den første med ham og vist ham hvordan han kunne bruge flowdiagrammet som sin vejledning.

4.5 Forløbet som helhed

Ser vi på forløbet som helhed, så kan man se at eleverne har været rundt i alle områder af Kolbs læringscirkel flere gange, men at størstedelen af tiden er blevet brugt i øverste venstre hjørne. Der har været stor vægt på praksislæring, og i nogen grad også divergent læring. I slutningen af forløbet er eleverne i stand til at anvende deres erfaringer til at indgå i en reflekterende, iterativ udviklingsproces.

Mange dele af forløbet følger Kolbs læringscirkel i sin fulde udfoldelse, men der er også segmenter hvor et eller flere stadier ikke er kommet i spil som sådan. Det er ikke nødvendigvis et problem – der er for eksempel ikke noget i vejen med at foretage flere eksperimenter og observationer, før man når til at begrebsliggøre det man har fundet ud af. Til gengæld kan

man overveje, om læringsudbyttet ville være større hvis abstrakt begrebsliggørelse oftere blev efterfulgt af udvidelse gennem eksperimenteren, snarere end at undervisningen går videre til et nyt koncept.

En elev fortalte til det efterfølgende interview at *'det har været det sjoveste forløb indtil videre'*. Dette matcher min observation af at eleverne har været engagerede under forløbet, og har nydt den legende tilgang til programmering. En anden sagde at *'det var nemt at forstå selvom man ikke er så god til programmering'*. Tanken var at tage hvad der for en erfaren programmør foregår som mentalt arbejde, og stilladsere det med fysiske artefakter, og denne udtalelse tyder på at det er lykkedes.

Udspurgt om hvorvidt de kunne finde på at bruge kortene til andre projekter, udtalte en elev: *'kun hvis vi ikke kan løse det uden, for det tager ret lang tid'*. Dette tyder på at eleven ser kortene som et værktøj der kan gøre det nemmere at konstruere programmer, men som ikke altid er besværet værd.

4.6 Efter forløbet

I et efterfølgende projektforløb har jeg en enkelt gang fundet kortspillet frem igen, da en gruppe elever sad med et problem de havde svært ved at overskue[1]. Så snart jeg foreslog dem at bruge kortene på samme måde som under algoritmeforløbet, fandt de selv ud af hvordan de kunne repræsentere problemet ved hjælp af kortene, og arbejde sig frem til en løsning.

5 Vurdering af forløbet

Alt i alt vil jeg mene at forløbet har været en succes. Det er lykkedes for mig som lærer at opstille nogle rammer, der gjorde det muligt for eleverne at arbejde induktivt og lære ved at eksperimentere og erfare. Eleverne har været engagerede og motiverede, og har vist tydelige tegn på læring. Ikke desto mindre er der nogle udfordringer jeg gerne vil løse, og nogle justeringer jeg gerne vil foretage, inden jeg kører forløbet igen.

En udfordring ved forløbet er at nogle øvelser afhænger af at de forskellige grupper er nået ca. lige langt – her tænker jeg på de øvelser hvor grupperne skal afprøve hinandens algoritmer. Der var tidspunkter under forløbet, hvor nogle grupper var færdige med deres produktioner, og måtte vente på at resten indhentede.

Et andet problem der opstod, var at ikke alle øvelser kunne engagere en gruppe på fem elever. Man kan overveje om det ville være bedre med lidt mindre grupper, på 3-4 elever i stedet for 4-5, eller om det ville give mening at skifte grupper undervejs, så man kan have store grupper til den første aktivitet, og lidt mindre grupper til den anden. Her er det også relevant at arbejde med Lave og Wengers koncepter om deltagelsesbaner og legitim perifer deltagelse[3, 10], for at sikre at alle elever har en tydelig og legitim deltagelsesbane.

Derudover kunne jeg godt tænke mig at tage segmentet om køretidskompleksitet og flytte det til sit eget forløb, da det ikke passede specielt godt ind. Øvelsen med flowdiagrammerne lægger fint op til en snak om køretidskompleksitet, men de efterfølgende segmenter brugte det ikke til noget – eleverne har ikke haft mulighed for at eksperimentere med begrebet og opnå yderligere erfaringer. Til gengæld kunne jeg så godt tænke mig at inkludere noget materiale omkring den iterative udviklingsproces, da jeg bemærkede at denne opstod meget naturligt i forløbet.

Den første og anden aktivitet foregik på forskellige dage, og da eleverne skulle i gang med den anden aktivitet, var de fleste af dem nød til at bruge en del tid på at genbesøge arbejdet fra den

[1] Gruppen var i gang med at implementere spillet *Skyscrapers*, og havde svært ved at gennemskue hvordan programmet skulle tjekke om en løsning var gyldig.

første aktivitet. En mulighed er at gøre dette til et eksplicit segment af undervisningen – der er argumenter for og imod. På den ene side var det en faldgrube for de af eleverne der forsøgte at gå i gang med det samme. På den anden side er der måske noget værdifuldt i selv at gøre sig den erfaring. Under alle omstændigheder skal forløbet tilpasses til skemalægningen – i dette tilfælde var jeg så heldig at få tre lange dage med tre undervisningstimer, og hvis man i stedet har fire eller fem undervisningsgange af to timer, så påvirker det også forløbets sammensætning, særligt i forhold til overgange mellem de forskellige segmenter.

Jeg havde en god oplevelse med at sammensætte eleverne baseret på fagligt niveau, i nogenlunde homogene grupper. Ideen med det var at de skulle eksperimentere sig frem til svarene, og det var også det der skete i timerne.

6 Konklusion

Formålet med opgaven var at undersøge, om læringsspil inspireret af unplugged computing kunne hjælpe eleverne med at gå fra at læse og forstå kode, til at skrive deres egen. Til det formål har jeg kørt et forløb bygget omkring netop sådanne aktiviteter, og analyseret det ved hjælp af Dewey og Kolbs teorier om erfaringslæring.

Baseret på min aktion og analyse, mener jeg at aktiviteter af denne type har stort potentiale til at understøtte elevernes arbejde med at skrive programmer fra bunden. Aktiviteterne giver eleverne mulighed for at eksperimentere med programmatisk tankegang på en engagerende og social måde, og de erfaringer de gør sig kan uden de store forhindringer anvendes i et programmeringssprog. Når eleverne først er bekendte med kort og spilleplader som metafor for variable og værdier i et program, kan de genbruge kortene i senere projekter. Kortspillet bliver altså til en fælles referenceramme som man kan referere tilbage til, når man skal minde eleverne om hvordan de kan finde på en algoritme der skal løse et problem. Endelig kan denne form for aktivitet tjene som udgangspunkt for undervisning om udviklingsprocesser, med fokus på iterativ udvikling.

Eleverne havde generelt en positiv oplevelse med forløbet, og stemningen i klasselokalet var god hele vejen igennem. Det var dog til tider kaotisk, og der var tidspunkter hvor ikke alle elever havde noget at lave, fordi nogle grupper blev hurtigere færdige med aktiviteterne end andre.

Litteratur

[1] S. Beck. *Didaktisk tænkning på arbejde*. Frydenlund, 2019. ISBN: 9788772162010.

[2] S. Beck, P. Kaspersen og M. Paulsen. *Klassisk og moderne læringsteori*. Hans Reitzel, 2014. Kap. 16. ISBN: 9788741258447.

[3] S. Beck og M. Paulsen. „Hvad er læringssamarbejde?“ I: *Gymnasiepædagogik* 86 (2011).

[4] Timothy Bell m.fl. „Computer Science Unplugged: school students doing real computing without computers“. I: *The New Zealand Journal of Applied Computing and Information Technology* 13 (jan. 2009).

[5] Jens Bennedsen og Michael E. Caspersen. „Exposing the Programming Process“. I: *Reflections on the Teaching of Programming: Methods and Implementations*. Red. af Jens Bennedsen, Michael E. Caspersen og Michael Kölling. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 6–16. ISBN: 978-3-540-77934-6. DOI: 10.1007/978-3-540-77934-6_2. URL: https://doi.org/10.1007/978-3-540-77934-6_2.

[6] Leonard Busuttil og Marquita Formosa. „Teaching Computing without Computers: Unplugged Computing as a Pedagogical Strategy“. I: (jan. 2020), s. 569–587. DOI: 10.15388/infedu.2020.25.

[7] Michael E. Caspersen og Palle Nowack. „Computational Thinking and Practice - A Generic Approach to Computing in Danish High Schools“. English. I: 2014.

[8] Teach London Computing. *The Teleporting Robot (and Melting Snowman) Activity*. 2016. URL: https://teachinglondoncomputing.org/resources/inspiring-unplugged-classroom-activities/the-teleporting-robot-activity/ (bes. 24.02.2022).

[9] D. Kolb. „Erfaringslæring – processen og det strukturelle grundlag“. I: *49 tekster om læring*. Red. af K. Illeris. Samfundslitteratur, 2012.

[10] K Nielsen. „Læring i et situeret perspektiv“. I: *Læringsteori & didaktik*. Red. af A Qvortrup og M Wiberg. Gyldendal, 2013.

[11] *Principles – CS Unplugged*. URL: https://www.csunplugged.org/en/principles/.

[12] S. Sentance, E. Barendsen og C. Schulte. *Computer Science Education: Perspectives on Teaching and Learning in School*. Bloomsbury Publishing, 2018. Kap. 9. ISBN: 9781350057128. URL: https://books.google.dk/books?id=v71KDwAAQBAJ.

[13] Undevisningsministeriet. *uvm.dk*. 2017. URL: https://www.uvm.dk/gymnasiale-uddannelser/fag-og-laereplaner/laereplaner-2017/valgfag-laereplaner-2017 (bes. 13.05.2022).

A Forløb om algoritmer

I marts 2022 kørte jeg et kort men koncentreret forløb om algoritmer med min 2.g-klasse i Programmering B. Under forløbet skulle eleverne opnå erfaring med at udvikle deres egne algoritmer, og i det hele taget få en bedre forståelse for algoritmebegrebet.

Til det formål arbejder vi med simple algoritmer ved hjælp af spillekort og terninger. En liste af tal kan repræsenteres af en række af spillekort, og en tælle-variabel kan repræsenteres af en 20-sidet terning. På den måde kan eleverne udføre en algoritme – for eksempel for at sortere listen af tal – ved at bytte rundt på kort og vende terninger efter nogle fastlagte regler. Vi bruger altså kortene og terningerne til at simulere hvad der sker i computerens hukommelse, mens en algoritme kører.

A.1 Kontekst for forløbet

Den 2.g-klasse jeg kører forløbet med, har på det tidspunkt haft programmeringsundervisning i lidt over et halvt år. I løbet af den tid har vi gennemgået en stor del af det valgte programmeringssprog (Python), og eleverne har lavet to større projekter. De er bekendte med variable, funktioner, forgreninger, lister, og løkker – de kender altså alle de grundlæggende byggeblokke, som bruges når man arbejder med algoritmer.

Jeg bedømmer at forløbet vil tage i omegnen af 10 moduler af 60 minutter at køre. Oprindeligt var tanken at forløbet skulle foregå over flere uger i løbet af marts og april, men pga. en aflyst studietur og medfølgende skemaændringer, får jeg en uge i starten af marts hvor jeg har 9 moduler med dem fordelt på tre dage, og vælger at placere algoritmeforløbet her.

A.2 FIMME

Formål

Eleverne skal øve sig på at løse problemer på en sådan måde at løsningen kan implementeres i et programmeringssprog, altså at formulere deres løsninger ved hjælp af de almindelige kontrolstrukturer (løkker, forgreninger).

Derudover skal de også kende forskellen på en *algoritme*, og en konkret *implementation* i et programmeringssprog.

Indhold

Algoritmer der arbejder med lister, bl.a. sorteringsalgoritmer. At finde på egne algoritmer. Flowdiagrammer og pseudokode som to måder at repræsentere algoritmer på. Implementation af simple liste-algoritmer i Python.

Derudover berøres perifert: Køretidskompleksitet, automatiserede tests.

Metode

Forløbet foregår over tre dage, med hver tre timers undervisning.

- Dag 1: Introduktion til forløbet og de algoritmer der skal arbejdes med
    1. Fravær og agenda
    2. Genopfriskning fra sidste forløb (kontrolstrukturer, som også skal bruges i algoritmeforløbet)

3. Intro til algoritmeforløbet
4. Definition af begrebet *algoritme*
    1. Bud fra eleverne
    2. Robot-eksempel[8]
    3. Formel definition: *En algoritme er en række trin man kan udføre for at løse en opgave, uden nødvendigvis at forstå hvorfor.*
    4. Diskussion i klassen: Hvorfor er det vigtigt at man ikke behøver at forstå hvorfor?
5. Vores første algoritme: Find det største tal

   Store kort med tal lægges ud i blandet rækkefølge, face down. To elever agerer variable – hver af dem må kigge på et kort ad gangen, og de må sammenligne og fortælle resten af klassen hvilket der er størst. Det er klassen der bestemmer hvilke kort de to elever samler op, og hvor og hvornår kortene ligges tilbage.

   Under disse regler får klassen nu til opgave at finde det største tal i listen. Håbet er at de finder ud af at sætte det i system, så de f.eks. starter fra den ene ende, og hele tiden ligger det mindste af de to tal tilbage.
6. Kort diskussion: Klassen har nu i fællesskab fundet på deres første algoritme. Kan vi lave nogle instruktioner, som andre ville kunne følge? Disse skrives på tavlen.
7. Flere algoritmer

   Eleverne inddeles i fire grupper, og hver gruppe får udleveret en beskrivelse af et problem, som kan løses med en algoritme.
    1. Problembeskrivelser, kort-lommer og blanke kort udleveres
    2. Reglerne gennemgås kort
    3. Elever arbejder med algoritmerne, og nedskriver instruktioner
    4. Efterhånden som hver gruppe bliver færdige, får de udleveret en problembeskrivelse mere
8. Afprøvning af instruktioner

   Grupperne får nu mulighed for at gå rundt og afprøve de instruktioner, som de andre grupper har lavet. Her vil de nok opdage en del tvetydigheder, der gør det svært at udføre algoritmerne.
9. Opsamling: vigtigheden af at lave præcise instruktioner, især når modtager ikke ved hvad algoritmen skal gøre!

- Dag 2: Spilleplader og flowdiagrammer

1. Fravær og agenda
2. Kort opsamling fra sidst: algoritmerne, pseudokode som instruktioner, vigtigheden af præcision
3. Flow-diagrammer, kort recap: Hvordan var det nu med de forskellige typer bokse? Hvordan repræsenterer man en løkke i et flow-diagram?
4. Spillepladerne introduceres, og vi spiller igennem FindMax i fællesskab.
5. Grupperne får hver udleveret en spilleplade til enten BubbleSort eller InsertionSort, samt tilhørende flow-diagram, som de spiller igennem. Derefter byttes der, så alle får prøvet begge. De har følgende opgave mens de spiller:
    1. Tæl hvor mange gange to tal bliver sammenlignet

2. Giv et gæt på hvad der sker med antallet af sammenligninger, hvis listen er dobbelt så lang

6. Diskussion om køretid baseret på optælning af sammenligninger – hvorfor er InsertionSort så meget hurtigere end BubbleSort? Kan det blive endnu hurtigere?
7. Grupperne udarbejder egne spilleplader og flow-diagrammer, baseret på en af de fire algoritmer fra sidst.
8. Grupperne afprøver hinandens spilleplader og flowdiagrammer. Virker de efter hensigten?
9. Opsamling: Er flowdiagrammerne nemmere at følge end den uformelle pseudokode fra sidst?

- Dag 3: Implementation i Python

1. Fravær og agenda
2. Tavlegennemgang: Hvordan oversætter vi mellem flowdiagram og kode? Særligt fokus på hvordan man kan spotte en løkke i et flowdiagram, når der nu ikke er nogen særlig bloktype for løkker.
3. Klassediskussion om korrekthed og tests.
4. Eleverne løser opgaver (individuelt)
    1. Oversæt FindMax til Python
    2. Oversæt (mindst) en af de fire algoritmer I selv har lavet til Python
    3. Oversæt (mindst) en af de to sorteringsalgoritmer til Python
    4. Skriv tests til alle algoritmerne.
5. Opsamling på forløbet: Vi kigger tilbage på vores definition af algoritmer, og sammenligner de tre forskellige måder vi har repræsenteret algoritmer på: Som pseudokode, som flow-diagram, og som Python-program.

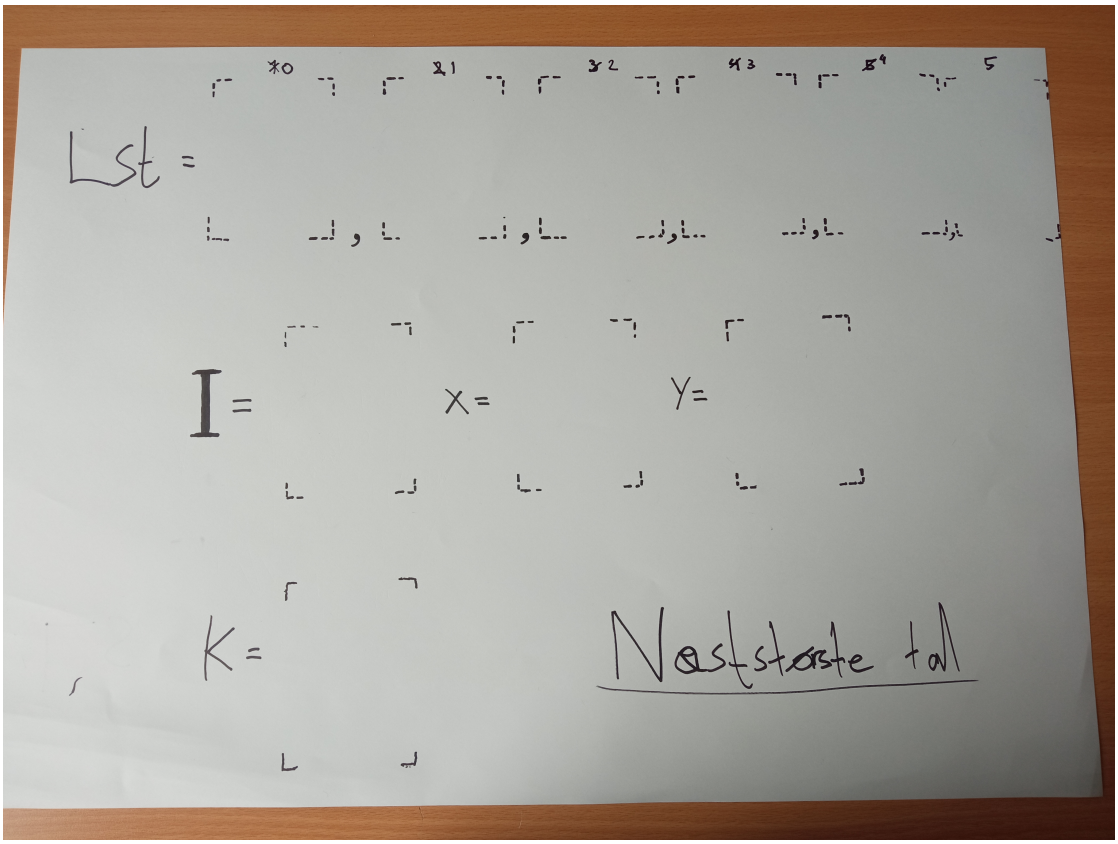
Materialer

- Slides med robot-eksempel
- Store kort med tal fra 1 til 10
- A2-papir og tuscher
- Spilleplader til Bubblesort og InsertionSort (to af hver), samt FindMax (bare en enkelt)
- Plastlommer til kort (magic-lommer, 100 stk)
- Blanke kort (100 stk)
- 20-sidede terninger (gerne spindowns, 5-6 stk)
- Flowdiagrammer for FindMax, Bubblesort og InsertionSort
- Beskrivelser af de fire problemer eleverne skal løse

Evaluering

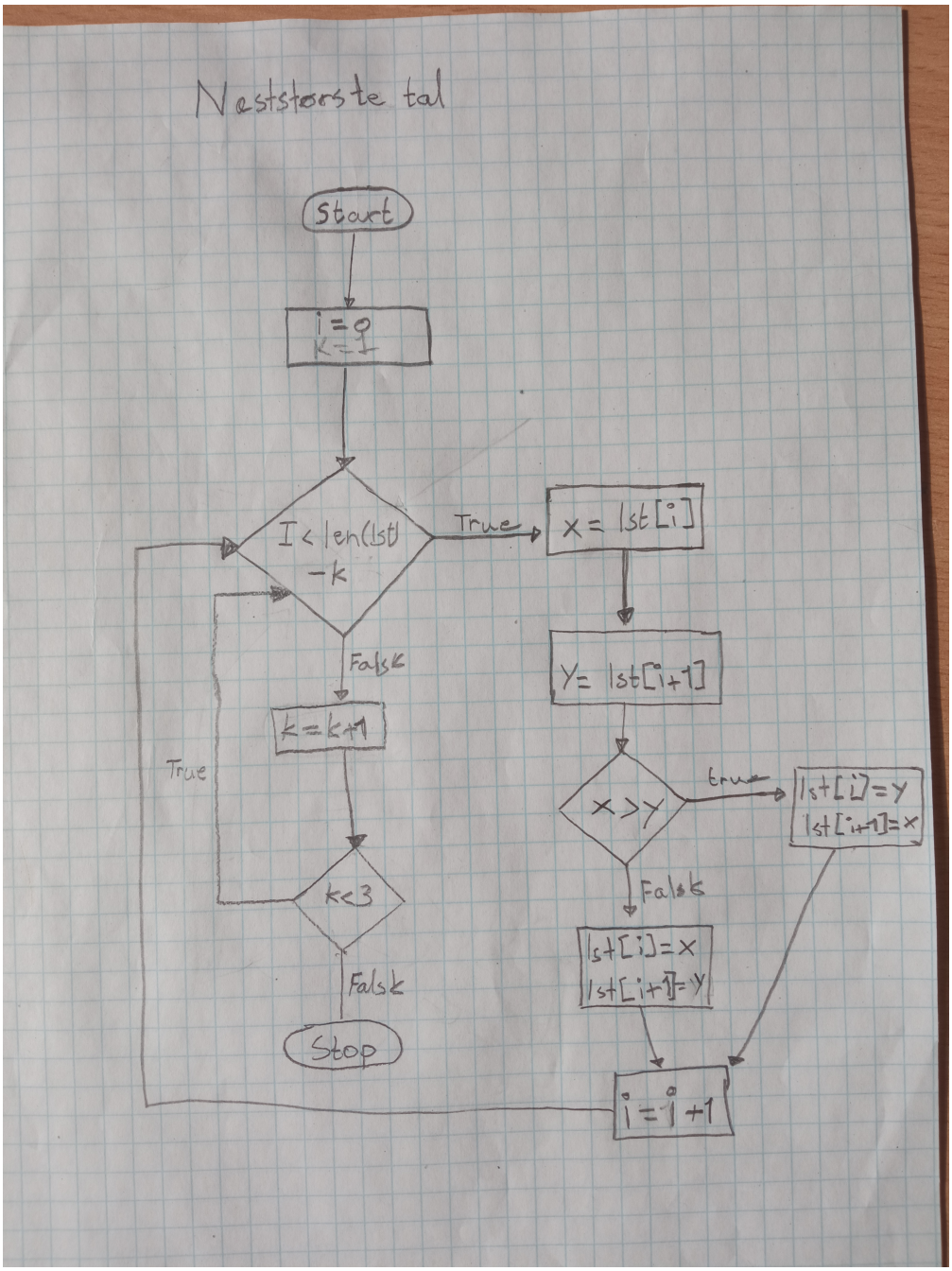
Der er ikke nogen skriftlig aflevering knyttet til forløbet, så evalueringen er primært baseret på observation af eleverne under forløbet. Særligt på sidstedagen holder jeg øje med hvordan det går med at implementere algoritmerne i programmeringssproget Python – kan eleverne overskue de indlejrede løkker og if-sætninger? Er det klart for dem hvordan sammenhængen mellem flow-diagram og kode skal være?

B Elevproduceret spilleplade



Spilleplade produceret af gruppe 3.

C Elevproduceret flowdiagram



Flowdiagram produceret af gruppe 3.