

Sudoku - Rapport

Ditte M. Nielsen, Anders B. Pedersen, Hjalte S. Sørensen
Hold 3, Gruppe 9, Lærer: Georg Strøm

10. juni 2008

Indhold

1. Indledning	3
2. Problemformulering og afgrænsning	3
3. Analyse af brugere og krav	4
3.1. Analyse af målgruppen	4
3.2. Analyse af problemstillingen	5
3.3. Teknisk analyse	6
3.4. Krav	6
3.5. Konklusion på analyse og krav	7
4. Design	8
4.1. Overordnet arkitektur	8
4.1.1. Valg af Java	8
4.1.2. Valg af MVC	9
4.2. Moduler	9
4.2.1. Game	10
4.2.2. View	10
4.2.3. GameController	11
4.3. Overvejelser omkring brugergrænseflade	11
4.3.1. Fysisk indhold	11
4.3.2. Interaktion mellem spiller og spil	13
4.4. Iterationer	15
4.5. Ændringer i forhold til endeligt design	16
4.6. Udeladte features og senere udvidelsesmuligheder	16
4.7. Konklusion på design	18

5. Implementering	19
5.1. Bogens metode	19
5.1.1. Sudoku-løser	19
5.1.2. Generering af plade	20
5.2. Vores implementering	20
5.3. Implementering af GUI	21
5.4. Dataflow	22
5.5. Konklusion på implementering	22
6. Test	23
6.1. Modultest	23
6.2. Systemtest	24
6.3. Brugertest	24
6.4. Konklusion på test	26
7. Konklusion	27
A. Appendix	28
A.1. Kravspecifikation	28
A.2. Baseline Design	33
A.3. Endeligt Design	45
A.4. Testspecifikation	56
A.5. Installations- og brugervejledning	71
A.6. Mødereferater	76

1. Indledning

Dette er en rapport over 1.årsprojektet på Datalogi. Emnet for vores projekt er sudokuspil til undervisningsbrug i 0.-3. klasse. Vi fandt sudokuprojektet meget interessant af flere årsager. For det første er vi matematikere og dermed interesserede i det matematiske aspekt af at løse sudokuer og de mekanismer, der ligger bag. For det andet synes vi, at det ville være rigtig spændende at beskæftige os med et program til børn, idet vi finder det meget interessant at gøre undervisningen mere tiltrækkende for børn vha. computerspil, og at iagttage den næsten magiske tiltrækningskraft, som computerspil har på børn. Til sidst fandt vi det også tiltrækkende, at dette projekt taler så klart til de færdigheder, vi opbyggede på kurset OOPD.

Rapporten er indholdsmæssigt delt op som foreslået på Absalon, dog er vores designafsnit, især beskrivelsen af GUT'et, forholdsvis langt, da vi gerne fyldestgørende vil gøre rede for de overvejelser og beslutninger, der er taget i forhold til at lave et computerspil til børn. Projektet vil kunne læses og forstås af enhver, der har gennemført kurset OOPD eller tilsvarende. Et kig i kursets tilhørende bog ("Software Engineering for Students", [3]) vil dog være en hjælp.

2. Problemformulering og afgrænsning

Formålet med dette projekt er, via vores undersøgelse af målgruppen, at skabe et computerbaseret sudokuspil, der kan bruges i undervisningen af børn i 0.-3. klasse i stedet for kopierede papirsudokuer. Det skal kunne generere tilfældige sudokuplader og yde hjælp til børnene, hvis de går i stå, så det på denne måde aflaster lærerne i forhold til at løse sudokuer på papir. For at lærernes arbejdsbyrde ikke forstørres unødigt, er det også vigtigt, at programmet er let at installere og afvikle. Desuden er det vigtigt, at det er nemt og overskueligt at bruge for børn på disse klassetrin, og at programmet samtidig kan være med til at lære dem logisk tankegang og give dem interesse for matematik.

Det forventes, at læreren forklarer reglerne for sudoku. Dette er altså ikke programmets opgave. Da funktionalitet og brugervenlighed er højt prioriteret i dette projekt, vil vi ligeledes ikke bruge megen tid på avanceret grafik og animationer til vores program. På grund af projektets begrænsede tidshorisont vil spillet blive udviklet som et selvstændigt, stationært program. Det vil sige, at det ikke vil være webbaseret og heller ikke med mulighed for at spille på et netværk. Derudover vil det ikke være muligt at gemme spil, man er i gang med, og det vil heller ikke være muligt for lærerne at tjekke, om eleverne rent faktisk har løst sudokuer, efter spillet er afsluttet.

3. Analyse af brugere og krav

Som indledning til vores projektforløb havde vi et møde med Georg, hvor han præciserede sine forventninger til det sudokuspil, vi skulle udvikle. Dette er hovedgrundlaget for vores projekt, men også to uformelle interviews med lærere har været med i vores overvejelser omkring design, samt lagt grundlag for nogle idéer og beslutninger. Vi har ikke i denne proces været i kontakt med børn før brugertesten af programmet, og dette er en svaghed i vores analyse, har vi senere indset, da det er børnene, der er de egentlige brugere. Dog mener vi, at denne svaghed ikke har afgørende betydning for projektet. Dette betyder, at nedenstående afsnit omkring målgruppen udelukkende er baseret på de voksnes idéer og forventninger.

3.1. Analyse af målgruppen

I dette projekt er målgruppen børn i SFO-alderen, det vil sige skoleelever i 0.-3.-klasse. Da de yngste børn således kun lige er begyndt i skole, må vi forvente, at deres talundskaber er begrænsede, men at de sandsynligvis kan tallene op til 10. Dette bekræftes af en af vores skolelærere:

”Eleverne arbejder med / lærer tallene i bhv.kl., men der er stadig nogle elever, som ikke er sikker i tallene over 5 i begyndelsen af 1.kl.”

Marianne Nielsen, folkeskolelærer.

Vores anden skolelærer, Anders Samunsen, mener, at børnene kan tallene fra skolestart. På denne baggrund konkluderer vi, at alle børnene vil kunne spille en 4x4 sudoku med tal.

Derimod er det helt sikkert ikke alle børnene, der har lært at læse endnu, men vi går ud fra, at de kender, eller meget hurtigt kan lære, de mest anvendte ord i computerdialoger som ”Ok”, ”Luk” med flere. Dette mener også både Georg og Marianne.

Vi forventer, at alle børnene har spillet computer før og dermed forstår, hvordan den basale interaktion med et computerspil foregår, samt hvordan forskellige handlinger som f.eks. ”drag and drop” virker.¹

Dog må vi regne med, at børnene ikke har det store overblik endnu, hvorfor vores brugergrænseflade skal være så simpel som mulig.

Programmet er primært til undervisningsbrug, så børnene sidder mange i samme rum og spiller spillet samtidig, og derfor må der ikke være elementer i spillet, der forstyrrer

¹Dette mener Georg også er rimeligt.

dem, der sidder ved siden af.

3.2. Analyse af problemstillingen

Sudoku (eller børnesudoku) bliver brugt i matematikundervisningen i indskolingen ved, at læreren uddeler kopier af sudokuer, som børnene så løser.

Der ønskes nu at dette indslag i undervisningen bliver gjort mere spændende i form af et computerbaseret sudokuspil, der kan erstatte sudokuløsning på papir. Dette har to hovedformål:

1. at gøre det lettere for læreren i undervisningen,
2. at gøre det mere spændende for børnene og dermed øge deres engagement.

Udfra afsnittet om målgruppen og Georgs kommentarer til det første møde kom vi frem til, at disse to punkter kan beskrives mere indgående på følgende måde:

Det første punkt indebærer bl.a., at læreren ikke skal kopiere sudokuer, idet programmet selv skal kunne generere forskellige valide sudokuer. Desuden skal det være muligt at vælge to størrelser, 4x4 og 9x9, så der på denne måde gerne skulle skabes en niveaudeling efter evner. Sidst men ikke mindst indebærer dette punkt, at programmet skal indeholde en hjælpefunktion, så børnene ikke behøver at få hjælp af læreren.

Dette er også en del af punkt to, idet børnene ikke så let går i helt stå, hvis de ikke selv kan komme videre. Derudover indeholder punkt to implicit, at der skal være en lettilgængelig brugergrænseflade, dvs. at den indeholder et minimum af tekst og at den muligvis har billeder og farver til at hjælpe med at forklare forskellige knappers funktioner. Brugergrænsefladen skal også indeholde et feedbacksystem og et pointsystem, der belønner børnene hen ad vejen og dermed motiverer dem, hvad man jo ikke kan forvente, at papirudgaven gør. Ligeledes vil de kunne motiveres af de to forskellige niveauer, så de kan prøve den svære, hvis de forbedrer sig. For de børn, der ikke er så gode til tal endnu, kunne endnu en motiverende faktor være, at kunne vælge at spille med symboler i stedet for tal.

Til sidst har vi overvejet, at der kan være i hvert fald to ulemper ved et computerbaseret sudokuspil: det kan være svært at installere, hvis man ikke ved noget om computere, og det kan køre for langsomt til at bevare børnenes koncentration. Derfor skal det være let at installere, og spillet skal have en meget lav responstid.

3.3. Teknisk analyse

Idet vores program i princippet skal kunne bruges på enhver skole, skal det kunne køre på flere forskellige platforme. Ifølge Georg skal det køre på lidt ældre PC'er og Mac'er. Dette uddyber vi til, at programmet skal kunne køre på alle Java 5-kompatible platforme.

3.4. Krav

Overvejelserne i de to ovenstående afsnit, 3.2 og 3.3, førte os til nedenstående 9 krav. Vi har her angivet vigtigheden af dem, idet dem med prioritet 1 er en del af baseline design, mens dem med prioritet 2 er en senere implementation. Desuden er det nævnt, hvilke eksterne kilder der er til kravene. Idet vi ikke nåede at tale med nogen lærere, før vi afleverede vores kravspecifikation, har vi skrevet lærerne på som kravstillere ud fra vores idéer om, hvad de ville ønske af programmet. Det er således også delvist vores egne krav.

- Algoritme til generering af valide 4x4 og 9x9 sudoku, Georg har stillet dette krav (prioritet 1).
- Simpel grafisk brugerflade, implicit krav som vi går ud fra, at en lærer ville stille (prioritet 1).
- Lav responstid, lærernes krav (prioritet 1).
- Nem installation og afvikling, herunder at spillet kan køres på de nævnte platforme, lærernes krav (prioritet 1).
- Der skal være et pointsystem, Georgs krav (prioritet 2).
- Hjælpesystem, Georgs krav (prioritet 2).
- Mulighed for flere sværhedsgrader, Georgs krav (prioritet 2).
- Mulighed for at spille med tal eller symboler, lærernes krav (prioritet 2).
- Der skal være et feedbacksystem, Georgs krav (prioritet 2).

Idet Georgs krav om hjælpesystem var ufravigeligt, burde det muligvis have været baseline, men i første iteration var der mere fokus på, at spillet virker, og at børnene kan finde ud af det.

For en nærmere beskrivelse af kravene henvises i øvrigt til kravspecifikationen i Appendix A.1.

3.5. Konklusion på analyse og krav

Ud fra mødet med Georg og interviews med lærerne kunne vi beskrive vores målgruppe som børn i indskolingen med computererfaring, der kan tallene men muligvis ikke kan læse. Denne viden samt de krav, Georg havde stillet fra begyndelsen, skabte vores kravspecifikation, hvor hovedpunkterne er, at det skal være et lettilgængeligt sudokuspil, der kan hjælpe børnene hvis de går i stå, og motivere dem via feedback og et pointsystem. Desuden skal det være let for lærerne at installere og afvikle det.

4. Design

Den helt overordnede struktur for vores design af sudoku-spillet bygger på Model-View-Control-mønsteret. I nærværende afsnit vil vi argumentere for valget af dette mønster, dets fordele og ulemper, valget af programmeringssprog og lignende. Vi vil desuden beskrive spillets enkelte moduler overordnet, deres indbyrdes samarbejde og afhængigheder, samt hvilke iterationer programmet er blevet udviklet i.

Herudover vil vi beskrive overvejelser i forhold til opfyldelse af kravspecifikationen (se Appendix A.1) samt gå i dybden med overvejelser omkring design af brugergrænsefladen og dennes funktionalitet, der spiller en meget væsentlig rolle i dette projekt.

4.1. Overordnet arkitektur

Helt overordnet har vi valgt at implementere spillet i Java ved hjælp af MVC-mønsteret. Ud fra dette er der lavet et baseline design indeholdende de krav fra kravspecifikationen med højeste prioritet, der som udgangspunkt *skulle* implementeres. Ud fra baseline designet er der videre lavet et udkast til et endeligt design ud fra en vurdering af tidshorizonten og gruppens kodekompetencer. Mere herom i afsnittet om iterationer.

4.1.1. Valg af Java

Vi valgte forholdsvist hurtigt at bruge Java som programmeringssprog. Dette var der flere grunde til:

- Java er platformsuafhængigt (i en vis forstand). Dette gør det lettere for os at distribuere spillet uden større hensyntagen til brugerens it-faciliteter, ligesom det muliggør distribution af spillet i én fil. Alternativt ville vi skulle kompilere programmet til forskellige platforme, ligesom mange proprietære programmeringssprog kun tillader afvikling på bestemte platforme.
- Java har et stort bibliotek af afprøvede og særdeles nyttige funktioner, specielt til design af en indbydende GUI (ved brug af komponenter fra Swing) og til interaktion mellem klasser.
- Dette projekt ligger efter vores mening i naturlig forlængelse af kurset OOPD på første år, hvor de samme teknikker blev anvendt. Vi så det derfor som en oplagt mulighed for at udvikle vores objektorienterede færdigheder ud over indholdet af OOPD-kurset.

- Java var det eneste sprog, som alle i gruppen før havde beskæftiget sig med. Det ville derfor have været umådelig tidskrævende, hvis dele af eller hele gruppen skulle sætte sig ind i et nyt sprog, inden udviklingen reelt kunne begynde.

4.1.2. Valg af MVC

MVC-mønsteret passer efter vores mening særdeles godt på projektet af flere årsager. For det første harmonerer det fint med valget af Java som udviklingssprog, da Java som udgangspunkt er objektorienteret og understøtter kommunikation mellem moduler i programmer. Blandt andet (og særlig relevant) understøttes idéen om at lytte på handlinger fra andre klasser.

Det er ligeledes ønskværdigt under udviklingen at kunne holde de forskellige komponenter i spillet så adskilt som muligt. F.eks. skal vi som udgangspunkt implementere en grafisk brugerflade (GUI), men MVC-mønsteret gør, at vi har mulighed for senere f.eks. at implementere flere udgaver af en sådan eller måske en tekstbaseret brugerflade (TUI).

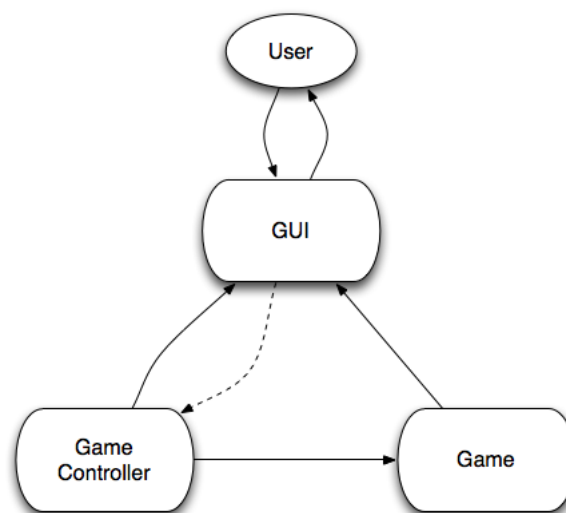
Der er undervejs i projektet foretaget visse afvigelser fra det generelle MVC-mønster. Disse beskrives nærmere senere.

4.2. Moduler

Programmet har vi valgt at inddele i 3 komponenter, der harmonerer med MVC-mønsteret: Game, View og GameController (svarende til hhv. Model, View og Control). Brugeren har udelukkende kontakt med View-modulet, resten er skjult. En sådan opdeling har undervejs også muliggjort en nemmere måde at fordele ansvar for udvikling, ligesom fokus i kritiske perioder kan rettes mod en bestemt komponent. Den overordnede struktur af programmet er vist i Figur 1 herunder.

I skemaet ses det, at brugeren (User) udelukkende interagerer med View-komponenten (vores GUI). Pilene på diagrammet skal symbolisere informationsflowet under afviklingen af programmet, snarere end afhængighed modulerne imellem.

Herunder følger en kort beskrivelse af de tre hovedmoduler i spillet: Game, View og GameController. En detaljeret beskrivelse af de enkelte modulers funktionalitet kan findes dels i afsnit 5 om implementering og i Appendix A.3 (Endeligt Design). Vi vil i dette afsnit ikke gå i dybden med programmets klassestruktur, da vi ikke finder dette relevant for rapportens overordnede indhold. Vi henviser i stedet til det endelige design i Appendix A.3.



Figur 1: Skitse af overordnet arkitektur i programmet

4.2.1. Game

Game indeholder selve spillet internt i programmet. Det er ligeledes kun denne komponent, der er i stand til at udføre konkrete ændringer på pladen, ligesom det også er her, at der kan genereres nye plader og afgøres, om sudokuen er løst. Altså indeholder modulet information om spillets status, dets videre forløb osv. Gamet får information om spillets forløb gennem controlleren og bliver herigennem også bedt om at udføre handlinger. Det er kun Gamet, der er i stand til at modificere direkte i selve spillepladen. Implementeringsmæssigt er en instans af Game også en repræsentation af en spilleplade. Mere herom i afsnittet om implementering.

4.2.2. View

View-modulet indeholder alle elementer i den grafiske brugerflade, dvs. de ting, som spilleren rent faktisk kan se og interagere med. Dette vil i vores tilfælde sige to vinduer: et opstartsvindue, hvor spilleren vælger sværhedsgrad, og hovedvinduet, indeholdende en spilleplade, mulighed for at vælge de tal, der skal indsættes, mulighed for at starte et nyt spil, mulighed for at få hjælp og mulighed for at afslutte spillet.

GUI'et interagerer direkte med GameControllern og henter data fra Gamet. Vi har valgt at gå bort fra det traditionelle mønster, hvor modellen underretter viewet om, at der er sket ændringer. Vi har i stedet valgt at lade controlleren håndtere brugerens input, videreformidle det til Gamet og derefter bede viewet om at opdatere. Grunden til dette

er dels, at der i et sudokuspil kun sker én ting på hele pladen af gangen (der indsættes eller fjernes et tal); derfor giver det ikke mening f.eks. at bede viewet om at opdatere hele pladen, hver gang der indsættes et enkelt tal (som man ellers normalt ville gøre med `notifyObservers()` i Java). Dels er der en del forskellige muligheder for, hvad der skal kunne ske, når modellen ændrer sig.

I View-modulet ligger ligeledes funktionalitet der vedrører feedback på udførte handlinger (dvs. de ting, der skal sættes i værk, når spilleren f.eks. har løst hele pladen eller blot en række/søjle/minigrid), samt dele af hjælpefunktionen og funktioner til tidsstyring.

4.2.3. GameController

Denne komponent styrer samspelet mellem de to andre komponenter, idet den sørger for at modtage input fra GUI-komponenten, afgør hvilken handling, der skal udføres på inputtet, og beder Game-komponenten om at udføre de spilhandling, der er nødvendige for at udføre det af brugeren ønskede træk eller programhandling. Herefter underrettes viewet om, hvorvidt der er sket en ændring og i så fald hvilken. GameController-modulet sørger ligeledes for at starte selve spillet ved at instantiere de rigtige klasser og lave forbindelser mellem dem.

4.3. Overvejelser omkring brugergrænseflade

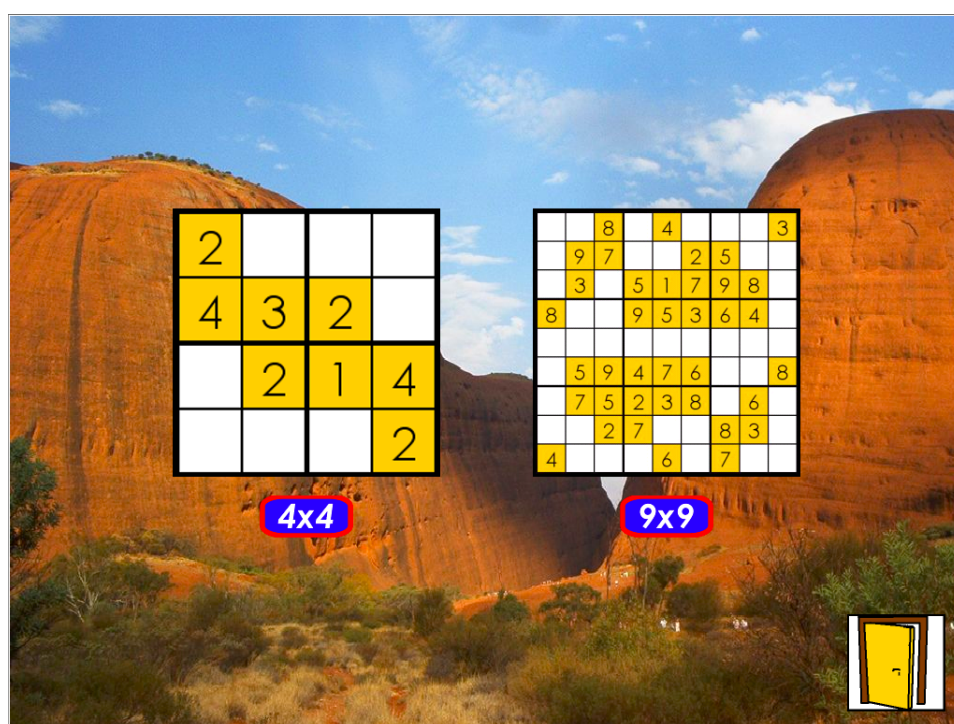
I analyseafsnittet ovenfor konkluderede vi, at programmet skal være et lettilgængeligt sudokuspil, der kan hjælpe børnene hvis de går i stå, og motivere dem via feedback og et pointsystem. Dette har ført til følgende konkretisering af indholdet og udseendet af brugergrænsefladen.

4.3.1. Fysisk indhold

Vores brugergrænseflade er forsøgt lavet med et så minimalt indhold som muligt, uden at kompromittere brugerens muligheder for at interagere med spillet på en tilfredsstillende måde. Mængden af tekst er begrænset til, hvad der står på tre af knapperne samt under selve spillepladen. Resten af interfacet domineres af tal, figurer og billeder, alt sammen i stærke klare farver og grupperet efter indbyrdes relationer. Der er to hovedkomponenter i brugerfladen: opstartsdialogen og selve spillepladen. Hele spillet er lavet til at køre i fuld skærm for at undgå forvirrende elementer som skrivebordet, andre kørende

programmer, menubjælker og lignende. Begge skærbilleder har et af gruppe-medlemmernes egne ferie-billeder som baggrund, og samtlige knapper og elementer i brugerfladen er hjemmelavede. Der er altså ikke anvendt ophavsretligt beskyttet materiale.

Vi begynder med opstartsdialogen, der kan ses nedenfor i Figur 2. Opstartsdialogen indeholder 3 muligheder: starte en ny 4x4-plade, starte en ny 9x9-plade og afslutte spillet. De to førstnævnte udføres ved et simpelt klik på enten pladen eller knappen repræsenterende den sværhedsgrad, man ønsker at spille. Sidstnævnte sker ved et klik på døren i nederste højre hjørne af skærmen (denne samt dens placering går igen på begge skærbilleder for at ensrette designet). Vælges én af de to første muligheder, bringes spilleren videre til det primære skærbillede (Figur 3).



Figur 2: Spillets indledende skærbillede (opstartsdialogen).

Som eneste gennemgående elementer er baggrundsbilledet og døren, der afslutter spillet. Spilleren præsenteres nu for selve spillepladen, som er enten 4x4 eller 9x9 midt på skærmen. Under pladen starter et ur, der tæller minutter og sekunder, siden spillet påbegyndtes, samt en tæller, der registrerer, hvor mange gange spilleren har fået hjælp undervejs.² Til højre for pladen findes de muligheder, brugeren har på spillepladen: 'Nyt spil', 'VINK', indsættelse af tal (1-4 eller 1-9), 'Slet tal' og afslut spillet (døren). Disse komponenter har vi anset som værende de mest nødvendige og vigtigste for at kunne

²Se eventuelt mødereferat fra tirsdag den 27. maj.



Figur 3: Spillets primære skærbillede (spillepladen).

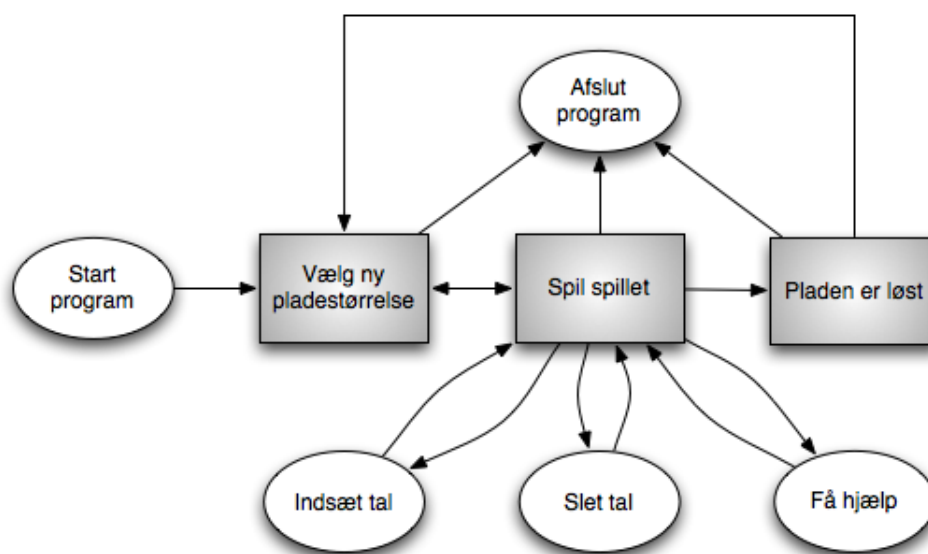
spille spillet tilfredsstillende. 'Nyt spil' og 'VINK' er forsynet med sigende symboler for at lette forståelsen af spillets muligheder for målgruppen. Ligeledes gør det det nemmere at navigere i spillet for børn, der endnu ikke kan læse. Desuden er 'Nyt spil'-knappen gjort grøn for at symbolisere noget nyt, ligesom 'VINK'- og 'Slet tal'-knapperne er holdt i samme blå farve som talknapperne for at signalere tilhørsforholdet hertil.

Der er anvendt samme skrifttype overalt i programmet (Century Gothic, sans-serif) i store, letlæselige størrelser. Desuden er spillet indrettet, så cursoren skifter til en hånd, hver gang musen føres over noget klikbart. Vi har bevidst heller ikke valgt at give spillet en bestemt titel, da det for det første er til undervisningsbrug og derfor ikke bør reklamere på nogen måde, ligesom de mindste børn alligevel ikke vil kunne læse titlen.

4.3.2. Interaktion mellem spiller og spil

Spillerens overordnede muligheder og forløb i spillet er skitseret i flowchartet (Figur 4) herunder.

Tal indsættes på pladen ved først at klikke på det pågældende tal til højre for pladen, og derefter på det felt på pladen, hvor det ønskes indsat. Gule felter på pladen kan ikke



Figur 4: Flowchart af brugerens mulige handlinger i spillet. Rektangler er forskellige skærm billeder; ovaler er handlinger.

ændres. Man sletter et tal på samme måde, ved enten at indsætte et andet tal på pladsen, eller klikke på 'Slet tal' og derefter på feltet. 'Drag and drop' er af tidsmæssige årsager blevet fravalgt, og vi har i egne tests og i vores brugertest fået bekræftet, at ovenstående metode virker efter hensigten.

To ting er værd at bemærke i spillet: feedback og hjælpesystemet. Spilleren får løbende feedback undervejs i spillet, og der er to typer af feedback: korrekt udfyldt række, søjle eller minigrid og korrekt udfyldt plade. Når brugeren har udfyldt en eller flere rækker/søjler/minigrids korrekt, blinker disse grønt et kort øjeblik og bliver derefter låst (farvet gule og kan ikke længere ændres). Vi har valgt at gøre dette dels som en hjælp til børnene, så de får positiv feedback på korrekt udførte handlinger, samt for at fastholde deres interesse omkring spillet i længere tid. Vi har bevidst valgt ikke at give feedback på hvert eneste korrekt eller ukorrekt indsatte tal, da vi mener, det ville give børnene for store muligheder for at snyde sig gennem spillet ved at prøve sig frem.

Hjælpesystemet fungerer på en forholdsvis intelligent måde. Hvis der på pladen er indsat forkerte tal, vil disse ved et klik på 'VINK'-knappen lyse rødt i et kort øjeblik. Hvis der ikke er nogen forkert indsatte tal på pladen, vises et nyt tal kort markeret med grønt, hvorefter det forsvinder fra pladen igen. Vi har ud fra samtaler med Georg og to skolelærere vurderet, at målgruppen godt vil kunne finde ud af denne tosidede funktionalitet i hjælpefunktionen. Hjælpefunktionen er ændret i forhold til det endelige design,

da forkerte tal her blev fjernet fra pladen, og det nye tal blev stående. Motivationen for denne ændring er nærmere beskrevet i testspecifikationen (Appendix A.4) og i afsnit 4.5.

4.4. Iterationer

Programmet er i store træk blevet udviklet i 3 iterationer. Iterationerne følger generelt kurssets afleveringer: baseline design, endeligt design, endelig kildekode. Forskellen mellem de to første er enorm, forskellene mellem de to sidste er af mindre karakter og til tider kosmetiske, men de er dog efter vores mening væsentlige at have med som en separat iteration.

1. iteration:

Er som udgangspunkt svarende til vores baseline-kode. Her er modellen oversat fra bogens kode (se senere) og gjort funktionsdygtig, men dog stadig fejlbehæftet. Der er håndkodet en midlertidig GUI kun bestående af almindelige knapper og en plade, ligeledes bestående af helt ens knapper. Funktionaliteten er begrænset til indsættelse af tal; der er endnu ikke implementeret tjek af korrekthed, hjælpefunktion, sletning af tal, feedback eller lignende. Man kan spille spillet, uden at vide, om det ens træk er korrekte, eller pladen er løst. 1. iteration opfylder dermed baseline-kravene i kravspecifikationen (se Appendix A.1), dog ikke på en specielt indbydende eller driftsikker måde.

2. iteration:

Svarer nogenlunde til det endelige design. Her er modellen fejlrettet og testet, ligesom den grafiske brugerflade har fået et kraftigt løft. Pladen er i farver efter felternes typer, knapperne har fået ikoner, der er to forskellige skærm billeder, feedback- og hjælpesystemet er på plads, og spillet kører i fuld skærm. Ligeledes er der et pseudo-pointsystem implementeret i form af en timer, der tæller, hvor lang tid spillet har varet. 2. iteration opfylder alle krav i kravspecifikationen (se Appendix A.1), på nær et egentligt pointsystem og muligheden for at spille med tal eller symboler. Begrundelser for dette kan læses i afsnit 4.6 nedenfor.

3. iteration:

3. iteration svarer til den afleverede kode. Denne iteration tager højde for de problemer og ændringsforslag, som udspringer af testspecifikationen og specielt brugertesten, som desværre først kunne udføres ret sent i forløbet. Indholdet af 3. iteration er nærmere beskrevet i afsnit 4.5 nedenfor. 3. iteration opfylder samme krav som 2. iteration.

4.5. Ændringer i forhold til endeligt design

I forhold til hvad der er beskrevet i det endelige design (se Appendix A.3), er der i 3. iteration (se ovenfor) sket små men væsentlige ændringer som følge af brugertesten:

- Hjælpefunktionen er blevet ændret, så den nu blot markerer forkerte eller nye korrekte tal et kort øjeblik. I det endelige design (og i 2. iteration) var den sat til hhv. at fjerne evt. forkerte tal eller at indsætte et nyt korrekt tal permanent. Motivationen for dette er, at vi under brugertesten (se afsnit 6.3) fik negativ respons på, at børnene ikke kunne se, hvad der var galt, når spillet blot fjernede de forkerte tal. Fra et pædagogisk synspunkt giver det også bedre mening at lade børnene selv regne ud, *hvorfor* et tal står forkert, i stedet for bare at fjerne det. På denne måde kan børnene heller ikke blot 'vinke' sig igennem et helt spil. Der er stadig en straf på 20 sekunder for at bruge 'VINK'-knappen. Vi havde overvejet at sætte denne straf op, men den nye udformning af hjælpefunktionaliteten kombineret med den tilføjede tæller for antal vink (se nedenfor) gjorde, at vi fandt tiden rimelig. Man kunne ligeledes også diskutere, om der overhovedet skulle være en tidsstraf.
- Der er blevet tilføjet et tæller for antal vink brugt i løbet af spillet. Denne kan sammen med timeren under pladen gøre det ud for et pseudo-pointsystem.

4.6. Udeladte features og senere udvidelsesmuligheder

Som beskrevet tidligere opfylder vores program ikke alle krav i den oprindelige kravspecifikation (se Appendix A.1). Herunder følger nu en liste over hvilke, og hvorfor de er udeladt:³

- *Pointsystem* (krav 4): Er delvist implementeret i form af en tidstager og en tæller for antal vink brugt i løbet af spillet. Af tidsmæssige årsager har vi måttet droppe en highscore-liste, men en sådan ville efter vores mening forholdsvis hurtigt kunne implementeres. Ulempen ved en sådan er dog, at de mindste børn, som ikke kan læse, måske ikke vil kunne forstå en eventuel dialogboks, der forklarer, at de er kommet på listen og ovenikøbet beder dem skrive deres navn. Dette strider også mod vores overordnede målsætning om, at hele spillet skal kunne gennemføres kun ved brug af en mus. Desuden vil det skabe endnu flere forskellige skærbilleder, men dette er et mindre problem.

³Se eventuelt mødereferater fra fredag den 23. maj og tirsdag den 27. maj.

- *Mulighed for flere sværhedsgrader* (krav 7): Dette krav er delvist implementeret i form af valgmuligheden mellem 4x4- og 9x9-plader i starten. Vores model er ligeledes gearret til at kunne håndtere forskellige sværhedsgrader inden for disse to pladestørrelser, men vi har af tidsmæssige årsager ikke nået at få det implementeret. Dette har også baggrund i, at det vil være et ekstra valg, børnene skal træffe, inden de kan komme i gang med selve spillet. Derfor har det ikke været prioriteret særlig højt.
- *Mulighed for at spille med tal eller symboler* (krav 8): Vi har valgt kun at tilbyde muligheden for at spille med tal. Dette valg er truffet på baggrund af samtaler med to skolelærere og er blevet bekræftet under brugertesten. De to skolelærere mente, at børnene kender tallene 1-5 stort set fra skolestart, og da vi ikke synes, der skulle symboler i 9x9 sudokuen, er der således ikke behov for at kunne spille med symboler. Valget er også begrundet i, at børnene skal kunne komme så hurtigt i gang med at spille som muligt. Endnu et ekstra valg, der skal træffes inden, vil efter vores mening forvirre unødigt og forkorte den effektive spilletid.

Vi har fra starten af projektet ligeledes valgt på forhånd at udelukke visse features, som vi enten fandt unødvendige for målgruppen eller for tidskrævende at implementere. Disse er blandt andet:

- *Større grad af feedback*: Den færdige implementation indeholder allerede et fungerende feedback-system. Dette kunne dog sagtens udvides med f.eks. lyde og animationer, specielt når en hel plade bliver løst. Hvis man tilføjer lyde, skal disse kunne slås fra på et vilkårligt tidspunkt i spillet.
- *Ikoner og grafik*: Eftersom ingen af gruppens medlemmer er grafiske designere (eller noget i nærheden), kan det overordnede look af programmet sagtens gøres pænere med f.eks. flottere knapper (måske endda animerede), en pænere ramme om spillepladen eller et andet baggrundsbillede. Desuden kunne man give spillet et sigende og fængende navn.
- *Mulighed for andre pladestørrelser og sværhedsgrader*: Som diskuteret tidligere, kan spillet udvides til at generere plader af forskellige sværhedsgrader inden for de to pladestørrelser. Vi har dog under konverteringen af koden forsøgt så vidt muligt at gøre funktionerne i modellen kompatible med pladestørrelser af et vilkårligt kvadrattal (n^2 , hvor $n \in \mathbb{N}$ og $n \neq 1$).
- *Netværksspil*: Sudoku i sig selv er ikke specielt godt egnet til at spille over netværk, men man kunne forestille sig et setup, hvor to spillere på forskellige geografiske

lokationer spillede den samme plade på tid mod hinanden. Ligeledes kunne en eventuel highscore-liste gøres global (eller lokal på f.eks. en skole).

- *Andre spillemetoder:* For de mere rutinerede spillere kunne det være en mulighed at optimere den tid og det antal klik, det tager at indsætte tal. Dette kunne f.eks. gøres vha. 'drag and drop' eller klik på et felt og efterfølgende tastning af tallet via computerens tastatur.
- *Udskriftsfunktionalitet:* Brugeren kunne gives mulighed for at udskrive den nuværende spilleplade, så den kunne tages med hjem eller på steder, hvor det ikke var muligt at anvende computer. Ligeledes kunne man forestille sig en klassesituation, hvor en lærer får mulighed for at udskrive et antal forskellige plader automatisk og i bestemte sværhedsgrader.
- *Mulighed for at gemme og åbne spil:* En oplagt mulighed ville være at kunne gemme et igangværende spil for derefter at kunne genoptage det senere. Dette ville også åbne op for portabilitet, idet man så kunne spille sit spil videre på en vilkårlig computer.

For en diskussion af fordele og ulemper ved det valgte design, henviser vi til Baseline Design (Appendix A.2) og Endeligt Design (Appendix A.3). En enkelt nævneværdig ting er dog, at vi fra starten burde have anvendt en form for Factory Method til vores design. Dette ville muliggøre, at man i fremtiden kunne udskifte vores GUI med en anden, eller f.eks. lave et alternativt feedback-system, der henvendte sig til forskellige aldersgrupper. Dette skulle være gjort fra starten i form af interfaces og abstrakte klasser i Java, der kunne ligge til grund for den videre udvikling. Dette opdagede vi dog for sent i processen, til at vi kunne nå at gøre noget ved det.

4.7. Konklusion på design

Vi har i det ovenstående beskrevet det overordnede og modulariserede design af vores sudokuspil, samt den grafiske brugerflade, der er et ganske væsentligt element i projektet. Vi har set, at designet overordnet følger MVC-mønsteret med få undtagelser, samt at vi har fået en efter vores mening tilfredsstillende opfyldelse af vores kravspecifikation. De mangelfulde steder har vi argumenteret for i de relevante afsnit.

5. Implementering

Implementeringen af programmet tager udgangspunkt i vores første baselinekrav, der siger, at vi skal lave en ny tilfældigt genereret spilleplade ved hvert nyt spil. Dette er den svære del af selve spillet at implementere, først at få genereret en valid sudokuplade, og dernæst at få fjernet tal fra den og checke, at det genererede sudokuspil har en unik løsning. Vi havde to ideer til løsningen af dette problem: enten at brute force hele vejen, hvor man så bliver nødt til at gå et skridt tilbage eller lave pladen helt om, hver gang noget går galt; eller at satse på logiske metoder til at finde tal at indsætte. Ud fra hvad vi har læst på internettet, kan begge metoder bruges til hurtigt at lave nye spil. Vi har lagt os fast på at bruge logiske metoder som udgangspunkt, og til dette har vi brugt Wei-Meng Lees "*Programming Sudoku*" [1], som beskriver en løsning til dette problem, samt hvordan det rent teknisk kan kodes i Visual Basic.⁴

5.1. Bogens metode

5.1.1. Sudoku-løser

Da stort set alle sudokuspil kan løses ved simpel logik, er første skridt i bogen at introducere en række logiske løsningemetoder til at finde tal at indsætte. Disse fungerer simpelt ved at gemme de mulige tal der kan stå i hvert felt, hvorefter metoderne checker disse muligheder i hver søjle, række og minigrid for at se, hvilke tal der med sikkerhed skal stå i et felt og/eller reducerer antallet af muligheder. Eksempler på disse metoder (fra Gameklassen) kunne være `checkColumnsAndRows()` eller `lookForTwinsInMinigrids()`.

Det er en klar fordel ved denne løsningsmetode, at hvis en af metoderne finder en værdi at indsætte, må det nødvendigvis være den værdi, der skal stå i feltet. Det er altså muligt at løse alle sudokuer, hvor man ikke skal gætte sig frem, kun ved brug af disse metoder. Pga. af vores forholdsvis snævert definerede målgruppe og dennes formåen har vi valgt kun at implementere op til og med bogens metoder med twins. Bogen har ligeledes også mere avancerede metoder (triplets), men vi har vurderet, at disse alligevel ville være for komplicerede for børn i målgruppen. Desuden anvender bogens fremgangsmåde også en brute force-teknik til at løse plader, der ikke nødvendigvis er entydige. Igen vil dette være over målgruppens niveau, så denne er kun brugt til generering af plader, ikke til løsning. Metoderne bruges nu også til at lave en metode, der kan løse sudokuer samt bedømme et spils sværhedsgrad ved at tælle, hvor mange og hvor avancerede løsningsmetoder der er brugt i processen.

⁴Se eventuelt mødereferater fra torsdag den 8. maj og mandag den 12. maj.

5.1.2. Generering af plade

Metoderne i sig selv er dog ikke nok til at udfylde en tom plade, da de kun indsætter tal, der med 100% sikkerhed skal være i et felt. Den udfyldte plade laves ved gentagent at indsætte et tilfældigt af de mulige tal, i det felt med færrest muligheder. Efter et tal er blevet indsat, prøves alle de logiske metoder, for at forsøge at indsætte tal, som nødvendigvis skal være i et felt. I denne proces bliver antallet af muligheder for hvert felt også opdateret. På denne måde vil metoden helt stoppe med at brute force, så snart sudokuen har en unik løsning. Kommer metoden til et felt, hvor intet kan indsættes, backtracks et skridt, hvorefter et nyt tilfældigt af de mulige tal indsættes.

Når pladen er udfyldt, laves selve spillet ved at fjerne et antal (bestemt af den ønskede sværhedsgrad) tilfældige felter. Til sidst køres sudoku-løseren for at bestemme den rent faktiske sværhedsgrad. Er pladen for svær at løse, byttes et sæt tomme felter ud for nogle andre, og pladen forsøges løst igen. Dette gentager sig et vist antal gange, hvorefter genereringen starter forfra, hvis resultatet ikke er tilfredsstillende.

5.2. Vores implementering

Implementeringen af sudoku-generatoren er primært en oversættelse af bogens metoder fra Visual Basic til Java, dog har vi ændret et par detaljer for at forbedre koden til vores behov.

Vi har ikke implementeret de mest avancerede løsningsmetoder. Da spillet er beregnet til børn i op til 3. klassetrin, er det muligt at generere tilpas svære sudokuer uden disse. De ville dog være nødvendige, hvis forskellige sværhedsgrader indenfor de enkelte pladestørrelser skulle implementeres. Til gengæld vil det ikke være svært at lave de ekstra metoder og få løseren til også at benytte disse. Herudover vil det generelt være nemt at implementere en mulighed for at vælge sværhedsgrad, da dette blot er et spørgsmål om, hvor mange og hvilke metoder løseren skal bruge for at løse en given plade. Samtidig er det at kunne vælge, at flere felter skal være tomme, allerede skrevet ind i koden men ikke brugt (`Point[] nineEmptyCells` og `Point[] fourEmptyCells` i `Game`-klassen).

I bogen sendes selve spillepladen ofte rundt mellem forskellige metoder som en lang tekststreng, hvor hver `char` angiver tallet på feltet. Efter kodeinspektionen blev vi dog gjort opmærksomme på, at dette kan medføre visse problemer. F.eks. sker der en masse konvertering af data frem og tilbage, hver gang tallene skal bruges, hvilket er ineffektivt. Desuden vil man i en eventuel udvidelse til sudokuer, der er større end 9x9, skulle tage højde for tocifrede tal, hvilket vil kræve unødigt meget plads samtidig med, at

en betydelig del af den eksisterende kode ville skulle ændres. Vi har derfor valgt at modificere de fleste af disse metoder, så de sender mere passende datastrukturer til hinanden, f.eks. `int`, `Point` eller `ArrayList<Point>`. På den måde vil hvert tal være repræsenteret af et `int`, så vi slipper for at konvertere data, og samtidig vil problemerne på det punkt være minimeret ved en udvidelse af spillepladen. Vores `possibleValues`-array er dog stadig holdt som `String`, da vi ikke har nået at konvertere alle de metoder, der anvender dette array.

Den genereringsmetode, vi har brugt, har dog den ulempe, at vi ikke kan bruge stacks på samme måde, som det gøres i bogen. Stacks bruges dog kun til backtracking, og en implementation af generatoren, der bruger de logiske metoder oftest muligt, gør, at der sjældent er brug for backtracking, så en simpel løsning på problemet var simpelthen at droppe backtracking og i stedet lave en helt ny plade, hvis generatoren ender med en plade, der ikke kan udfyldes. Vi har ikke oplevet, at dette var noget problem rent hastighedsmæssigt.

5.3. Implementering af GUI

Vi forsøgte i begyndelsen at programmere vores GUI vha. NetBeans' indbyggede GUI-modelleringsværktøj, men dette viste sig at være for besværligt. Grundet vores begrænsede erfaring med GUI, var NetBeans' GUI-generator mere til gene end en egentlig hjælp. I generatoren skal man vide præcist, hvor man kan ændre alting, hvilket besværliggør det at bruge hints til løsninger af problemer, som f.eks. fra Java Tutorials. Derfor har vi valgt at hardcode GUI'et, hvilket har givet os en større fleksibilitet og mulighed for at ændre småting undervejs.⁵

De to skærmbilleder i GUI'et er hver styret af en klasse, der står for at lave en `JFrame` og tilføje alle tilhørende elementer. Hver gang, der skiftes billede, smides den gamle frame væk, og en ny oprettes. Dette betyder, at der ikke er noget fremme på skærmen, mens den nye ramme oprettes. Vi har sent i processen indset, at det kunne have været smart kun at have én frame med én baggrund, som så kunne tilføjes et panel med de aktuelle elementer. På denne måde ville der hele tiden være et (og det samme) skærmbillede fremme. Det har dog ikke skabt nogen problemer og er ikke blevet bemærket under brugertesten, så vi har valgt at bruge vores tid på at ændre andre mere givende detaljer.

Når programmet køres på en computer, hvor full screen mode ikke er understøttet, har vi gjort en mindre fejl, da vores `initialDialog` i dette tilfælde har en ramme, men kan `resizes`, hvorimod `boardVieweren` ikke kan `resizes` og ikke har en ramme. Begge skulle

⁵Se eventuelt mødereferat fra fredag den 23. maj.

have haft en ramme og ikke kunne `resizes`, så vi har nogenlunde styr på elementernes placering.

5.4. Dataflow

Som nævnt i designafsnittet har vi ingen observers, og det er controlleren, der kalder relevante metoder for enhver action. Controlleren er tilføjet som `ActionListener` på 'Nyt spil'-knappen, 'VINK'-knappen og døren, som afslutter programmet. `BoardVieweren` overvåger selv talknapperne og holder styr på, hvilket tal, der er valgt, og kalder så Controlleren, hvis der bliver indsat eller slettet et tal på pladen.

Selve felterne på spillepladen er derimod lavet som `JLabels` og kan ikke have en `ActionListener`. De har i stedet `BoardVieweren` tilføjet som `MouseListener`. Hver gang der trykkes på et felt, registreres det altså af `BoardVieweren`, som så kalder Controllerens `cellPressed()`-metode. `cellPressed()` forsøger, hvis en af talknapperne er toggled, at indsætte værdien i det felt, der blev trykket på, både i `Gamet` (ved at kalde `Gamets assignValue()`-metode, der indsætter det valgte tal på den pågældende plads i `Gamets actual-array`) samt på spillebrættet (ved at kalde `BoardViewerens setCell()`-metode, der ændrer teksten på den pågældende `JLabel`).

Dernæst checkes, om henholdsvis hele spillebrættet, eller feltets række, søjle og minigridd er blevet korrekt løst, og i så fald kaldes en metode i feedback-klassen, der kort markerer det, der nu er korrekt. I det tilfælde at hele brættet er løst stoppes spillets timer, og et større disco-show iværksættes.

5.5. Konklusion på implementering

Vi har anvendt en metode til at generere spilleplader, som bygger på gennemtænkte logiske metoder. I forhold til alternativet (brute force) bruger vi måske mere tid på at finde hvert enkelt tal, men sandsynligheden for at skulle gøre ting om, er meget lille.

Vi har ud fra ideer om, hvad der fungerer bedst i vores program, tilpasset de metoder fra bogen, vi er blevet inspireret af.

GUI'et fungerer efter hensigten men har plads til småforbedringer.

6. Test

Til at undersøge om vores program rent faktisk virker, og om det opfylder vores krav, har vi benyttet forskellige strategier. Hovedsageligt har det været blackbox testing, idet vi ikke har lagt vægt på at teste metoderne i koden, men derimod funktionaliteten af programmet. Dog har vi testet enkelte af metoderne direkte, fordi de har afgørende betydning for nogle af vores krav. Resten af metoderne i koden anser vi for implicit testede gennem de andre tests, der udføres. Desuden ville det være meget besværligt at teste hver eneste metode, uden at være nødt til at bruge ikke-testede metoder til det, og vi ville være nødt til at gøre nogle metoder mere synlige (public), hvilket vi ikke ønsker i vores program. Dog har der i hele forløbet selvfølgelig foregået en slags løbende metode- og funktionstest, idet vi har afprøvet programmet uformelt, hver gang nye elementer blev tilføjet.

Formelt har vi sørget for at teste alle kravene mindst én gang, herunder undersøgt om spillet kan køres på Windows Vista og XP, Linux KDE og Gnome (OpenSuse 10.3) samt MacOS 10.5. I kravspecifikationen har vi skrevet, at spillet skal kunne køre på enhver Java 5-kompatibel platform, og med disse tests mener vi, at vi kan antage, at spillet kan køres på ikke-testede udgaver af styresystemer, som opfylder kravet om Java 5-kompatibilitet. Det ville heller ikke være praktisk muligt at teste på alle styresystemer, selvom det ville have været at foretrække at dække nogle flere, hvis tiden og vores ressourcer havde været til det.

De udførte tests består af en modultest, hvor der som før nævnt er testet vigtige metoder, en systemtest, hvor alle GUI'ets funktioner er testet, og en brugertest. Disse tests er nærmere beskrevet herunder. For nærmere beskrivelser af udførsel og resultat henviser vi til Testspecifikationen i Appendix [A.4](#).⁶

6.1. Modultest

I denne test har vi undersøgt følgende krav (se evt. Appendix [A.1](#)):

- Krav 1: Generering af valide 4x4- og 9x9-sudokuplader.
- Krav 3: Lav responstid.
- Krav 5: Hjælpesystem.
- Krav 7: Mulighed for flere sværhedsgrader.

⁶Se eventuelt mødereferat fra søndag den 1. juni.

Dette er gjort via en JUnit-test, som er implementeret i programkoden. Alle krav blev testet for både 4x4- og 9x9-sudokuer med det ønskede resultat.

Vi har ikke implementeret muligheden for flere sværhedsgrader ud over, at man kan vælge mellem 4x4- og 9x9-sudokuer, men i denne form anser vi dette krav for at være testet via de andre tests.

6.2. Systemtest

Her afprøves om funktionerne i GUI'et fungerer, som de skal. Det vil sige, at denne test udføres ud fra blackbox teorien, altså at vi tester funktionaliteten af programmet uden at tage hensyn til, hvordan det er implementeret. Denne del af vores test er meget vigtig, idet vores brugere ikke har adgang til koden, og det dermed gerne skulle påvises, at alting fungerer, når man kører spillet; uden at tage hensyn til implementationen af det. Det bemærkes, at de ændringer, vi indførte i 3. iteration efter brugertesten, også er afprøvet i denne test.

Resultatet vil også implicit afspejle om View-modulets samarbejde med Control-modulet fungerer, som det skal.

Systemtesten viste (jvf. Testspecifikationen Appendix [A.4](#)) at alle funktionerne i GUI'et virkede som de skulle, pånær at vi i Linux Gnome, en af de gange der blev klikket på 9x9-sudokuen i opstartsvinduet, oplevede, at næste vindue ikke kom frem. Dette regner vi dog for en sporadisk fejl.

Idet vi afprøvede alle GUI'ets funktioner, har vi også set, at følgende krav er opfyldt

- Krav 4: Pointsystem.
- Krav 5: Hjælpesystem.
- Krav 9: Feedback.

Altså fandt vi en enkelt fejl, som vi ikke kan begrunde, men udover det viste testen, at programmet reagerer som forventet på spillerens handlinger.

6.3. Brugertest

Brugertesten er både en afprøvning af, om programmet rent faktisk kan benyttes af brugere uden insiderviden om, hvordan alting burde fungere og en afprøvning af to af vores krav, der mønster sig på brugerens opfattelse af dele af spilprocessen. Det vil sige,

at det bliver en implicit afprøvning af de fleste af vores krav og en explicit afprøvning af følgende to krav:

- Krav 2: simpel grafisk brugerflade
- Krav 6: nem installation og afvikling.

Dermed bliver denne test på sin vis den vigtigste del af afprøvningen af programmet.

Idet vores program er et computerspil til undervisningsbrug, blev vi enige om, at den bedste måde at brugerteste det på, ville være at tage ud på en skole eller SFO og lade nogle børn spille det uden anden instruktion end sudokureglerne. Det ville ikke blive et rigtigt ”tænke højt forsøg”, for det mener vi ikke, at børn i den alder er i stand til, og i øvrigt er det også meningen, at lærerne kan instruere dem en smule i, hvordan spillet fungerer, hvis det bliver nødvendigt.

Selve forløbet af brugertesten foregik på Mariendal Friskoles SFO og tog en times tid. Vi havde otte børn til at hjælpe os, en fra 1. klasse og syv fra 2. klasse. Vi ville gerne have haft pædagogerne til at forsøge at installere programmet vha. vores installationsvejledning, men de havde desværre ikke tid, så vi installerede det selv uden problemer ved at kopiere filen Sudoku.jar fra en cd og over til computerne. Alle computerne havde Java Runtime Environment installeret, så spillet kunne køres med det samme. Dermed anser vi krav nummer 6 som testet, hvad angår installationen. Derefter fik vi børnene udleveret. De havde allesammen prøvet at spille sudoku før, så vi fortalte dem bare, at sudokuspillet lå på skrivebordet på computerne, og at de skulle begynde med den nemme. Ingen havde problemer med at starte programmet, hvorfor vi konkluderer, at den sidste del af krav nummer 6 er testet.

Derudover viste testen, at der var nogle ting, der kunne forbedres, nemlig⁷

- Klik med musen registreres ikke altid første gang i begge vores vinduer.
- 4x4 pladerne var for nemme.
- Feedbacksystemet blev i nogle tilfælde misforstået i 4x4-sudokuen, idet et tal ikke altid bliver gult, hvis det står rigtigt. Dette sker kun, hvis en række/søjle/minigridd er løst.
- Det lykkedes nogle at indsætte tal, uden at der var en aktiveret tal-knap.
- Timeren blev først opdaget sent i forløbet.

⁷Se eventuelt mødereferat fra mandag den 2. juni.

- 'VINK'-knappens funktion tog lidt lang tid at forstå, og der blev foreslået nogle ændringer til dens funktion.

Disse ting blev alle udbedret jvf. Appendix A.4, dog er den nye version ikke brugertestet. Til sidst skal det nævnes, at flere af børnenes umiddelbare forsøg på at indsætte tal var ved "drag and drop" -metoden, så der gik lidt tid, før de forstod, hvordan det skulle foregå. Desuden blev der efterspurgt flere sværhedsgrader, fordi 4x4 var for nemt og 9x9 for svært. Vi har af den grund og for at afhjælpe problemet i det tredje punkt ovenfor sat sværhedsgraden lidt op på 4x4-pladerne.

Vi anser vores brugertest for succesfuld, idet vi fik konstateret to fejl i programmet (at klik ikke blev registreret og at kunne indsætte tal, uden der var en aktiveret tal-knap), som vi fik rettet, og fordi vi efterfølgende kunne implementere nogle forbedringer. Grundlæggende virkede alt andet, som det skulle, og børnene synes, at det var bedre end at løse papirsudoku, især fordi de kunne få vink af hjælpesystemet, som ønsket. Dermed mener vi også, at kravet om en simpel grafisk brugerflade er testet. Det vil sige, at de to krav, der manglede at blive testet, er dækket af denne test.

6.4. Konklusion på test

Testfasen af vores projekt viste, at alle vores krav (pånær krav 8 om symboler, som vi ikke har implementeret, se afsnit 4 om Design) er opfyldte, og den afslørede nogle fejl, som vi fik rettet, før den endelige programkode blev afleveret. Desuden indeholder brugertesten idéer til yderligere forbedringer/udvidelser af programmet, som vi desværre ikke har haft tid til at implementere. Vi fik desværre ikke testet, om lærere eller pædagoger kan installere vores program, men idet vi i brugertesten uden problemer selv kopierede filerne og kørte programmet, går vi ikke ud fra, at der vil være problemer med dette. Ligeledes havde det været at foretrække at afprøve spillet på en gruppe af børn, der ikke alle havde prøvet at løse sudoku før, men da vores program alligevel forudsætter, at lærerne sætter børnene ind i det, synes vi alt i alt, at vores test var dækkende og med et godt resultat, der viser, at vores program fungerer som ønsket.

7. Konklusion

Vi har i dette projekt fået fremstillet og testet et sudokuspil, der efter vores mening opfylder kravene i problemformuleringen. Spillet er let at installere, er brugervenligt for børn i målgruppen og kan generere tilfældige sudokuplader i varierende sværhedsgrader.

Der er undervejs fraveget få krav helt eller delvist fra kravspecifikationen. Dette drejer sig om pointsystemet, muligheden for at spille med symboler og muligheden for flere sværhedsgrader på de enkelte pladestørrelser. Disse krav er blevet diskuteret i analyse- og designafsnittene, og er hovedsageligt blevet fraveget pga. manglende tid til implementering eller pga. samtaler med skolelærere om børnenes kompetencer.

Set i et retrospektivt perspektiv var der flere punkter, der kunne være håndteret anderledes undervejs i projektet. Blandt andet ville det have været en fordel at have undersøgt målgruppen nøjere, inden vi gik i gang med projektet, hvilket vi opdagede under brugertesten. Desuden kunne vi i gruppen måske have haft et bedre overblik over tidshorisonten og de ting, man kunne nå at lave inden for 7 uger.

Til sidst skal det dog nævnes, at programmet er testet for væsentlige fejl og mangler, ligesom vores brugertest har bekræftet os i vores analyse af problemet og de mulige løsninger hertil. Den testede målgruppe fandt desuden spillet spændende og interessant, så alt i alt kan vi konkludere, at projektet er lykkedes efter hensigten.

Litteratur

- [1] Lee, Wei-Meng: *Programming Sudoku*, APress USA (2006), ISBN: 978-1-59059-662-3.
- [2] Goldberg, Corey: *StopWatch.java* — *Java Timer Class*, <http://www.goldb.org/stopwatchjava.html>, hentet: 27. maj 2008, klokken 11:38:07.
- [3] Bell, Douglas: *Software Engineering for Students - A Programming Approach*, Fourth Edition, Addison-Wesley UK 1987 (2005), ISBN: 978-0-32126-127-4.

A. Appendix

A.1. Kravspecifikation

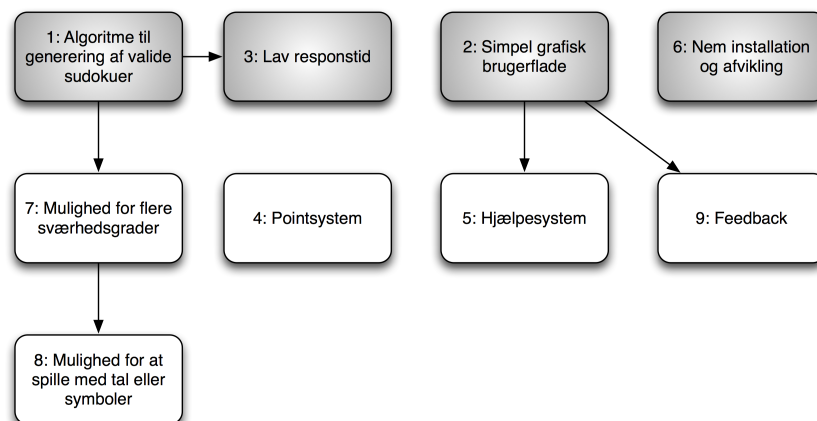
Baggrund:

I undervisningen i folkeskolen benyttes sudoku ofte i matematiktimerne. Der udleveres som regel kopier af sudokuer til eleverne, som de så løser. Hvis børnene går i stå skal læreren kunne hjælpe og have tid til at hjælpe dem hver især.

For at gøre dette nemmere ønskes der et sudokuprogram til brug i undervisningen, der genererer almindelige og børne-sudokuer ud fra en algoritme.

Formål:

For at gøre undervisningen med sudokuer bedre skabes et program, der er let og sjovt at bruge for børn i 0.-3. klasse, og som indeholder funktioner, der mindsker børnenes krav om lærerens hjælp. Det vil sige, at det skal have en grafisk brugerflade som børnene synes er underholdende, samtidig med, at børnene skal kunne bruge spillet uden at kunne læse. Spillet skal også indeholde en hjælpefunktion, der hjælper børnene videre, når de er gået i stå, og der skal være forskellige sværhedsgrader af spillet, så børn med forskellige kompetencer kan bruge det og forbedre sig undervejs.



Figur 5: Skitse af kravspecifikation. Grå er baseline, hvid er senere implementation.

Herunder er beskrevet de krav, der er stillet til softwaren. Hvis det er angivet, at det er et krav fra lærernes side, er det fordi vi går ud fra, at lærere ville ønske at disse ting var opfyldt for vores software.

1: Algoritme til generering af valide sudokuer

- Beskrivelse: Spillet skal selv kunne generere gyldige sudoku-plader i to størrelser: 4x4 og 9x9.
- Eksempel: Når der startes et nyt spil, præsenterer spillet med overvejende stor sandsynlighed en plade, som barnet ikke har spillet før.
- Begrundelse: I modsætning til et endeligt antal foruddefinerede plader, giver dette krav større fleksibilitet og flere nye udfordringer.
- Kilder: Eksternt (Georg) og internt.
- Prioritet: Baseline.
- Subkrav: 3, 7
- Superkrav: Ingen.

2: Simpel grafisk brugerflade

- Beskrivelse: Spillet skal kunne bruges via en simpel grafisk brugerflade, der uden yderligere hjælp kan anvendes af børn i 0.-3. klassetrin. I ordet simpel ligger følgende krav: Færrest mulige elementer i brugerfladen (knapper, menuer o.lign.), mindst mulig tekst.
- Eksempel: I princippet bør det være nok med følgende indhold af skærbilledet: selve sudoku-pladen og 3 knapper (nyt spil, hjælp og slut). Det tilstræbes desuden, at der er mulighed for at køre programmet i fuld skærm, hvorved resten af de forstyrrende elementer i styresystemet forsvinder.
- Begrundelse: Det er begrænset, hvad børn i målgruppen kan overskue af muligheder og information. De fleste vil heller ikke kunne læse selv simple instruktioner.
- Kilder: Eksternt (Lærerne).
- Prioritet: Baseline.
- Subkrav: 5, 9
- Superkrav: Ingen.

3: Lav responstid

- Beskrivelse: Programmets responstid holdes under 2 sekunder ved alle handlinger, på nær opstart.
- Eksempel: Der kan genereres en ny vilkårlig spilleplade inden for 2 sekunder fra klikket.

- Begrundelse: For at fastholde børnenes koncentration er det vigtigt, at de ikke er tvunget til at vente unødigt lang tid på computeren.
- Kilder: Eksternt (Lærerne).
- Prioritet: Baseline.
- Subkrav: Ingen.
- Superkrav: 1

4: Pointsystem

- Beskrivelse: Der skal være mulighed for konkurrence børnene imellem i form af antal opnåede point i spillet og en high-score liste. Pointsystemet skal tage højde for, om børnene undervejs har fået hjælp.
- Eksempel: Når der startes et nyt spil, starter en timer, der tæller f.eks. antal sekunder op, så længe spillet ikke er afsluttet. Anvendes hjælpesystemet undervejs, lægges der for hver gang point til timeren. De 10 bedste resultater registreres i en high-score liste med navn og pointtal. Der oprettes én liste for hvert niveau.
- Begrundelse: Konkurrence mellem børn øger lysten til at spille flere gange og at lære og forbedre sig, ligesom konceptet med "at kunne vinde" er endnu en motiverende faktor.
- Kilder: Eksternt (Georg).
- Prioritet: Senere implementation.
- Subkrav: Ingen.
- Superkrav: Ingen

5: Hjælpesystem

- Beskrivelse: Brugeren skal på passende tidspunkter i løbet af spillet kunne anvende en simpel hjælpefunktion (som kan medføre en straf i et eventuelt pointsystem).
- Eksempel: En hjælpeknop i spillet kan aktiveres, hvilket medfører én af to ting: hvis der er forkert placerede tal/symboler, vises disse fremhævet med f.eks. rødt i 2-3 sekunder; hvis der ikke er forkert placerede tal/symboler, vises med f.eks. grøn baggrund et nyt korrekt placeret tal i 2-3 sekunder. I begge tilfælde forsvinder markeringerne og/eller tallet igen efter de 2-3 sekunder.
- Begrundelse: Børnene kan på denne måde komme videre i spil, som de umiddelbart er gået i stå i, ligesom de har mulighed for at lære selve spillets regler at kende bedre.
- Kilder: Eksternt (Georg).

- Prioritet: Senere implementation.
- Subkrav: Ingen.
- Superkrav: 2

6: Nem installation og afvikling

- Beskrivelse: Lærere og forældre skal selv kunne installere og afvikle spillet, ligesom børnene selv skal kunne starte og afvikle spillet. Spillet udvikles i Java og skal kunne installeres og afvikles på enhver Java 5-kompatibel platform.
- Eksempel: Spillet distribueres som én fil, som kan distribueres på cd, via nettet eller lignende. Spillet kan køres ved blot at køre filen; i modsat fald fremgår det af brugsvejledningen, hvordan Java kan installeres på brugerens computer.
- Begrundelse: Vi antager, at stort set alle folkeskoler er i besiddelse af én eller flere Java-kompatible platforme. Desuden vil ovenstående sikre, at skoler og forældre skal bruge mindst mulige ressourcer på installation og afvikling af spillet.
- Kilder: Eksternt (Lærerne).
- Prioritet: Baseline.
- Subkrav: Ingen.
- Superkrav: Ingen.

7: Mulighed for flere sværhedsgrader

- Beskrivelse: Brugeren gives mulighed for at vælge mellem et antal forskellige sværhedsgrader.
- Eksempel: Brugeren præsenteres først for valgmuligheden mellem pladestørrelserne 4x4 og 9x9. Herefter vælges sværhedsgraden på et nyt skærbillede. Dette kunne eksempelvis bestå i forskelle i antallet af forudfyldte felter eller placeringen af disse på pladen.
- Begrundelse: Sværhedsgrader er med til at skabe progression og dermed fastholde spillerens interesse i spillet; spillets levetid forlænges herved også.
- Kilder: Eksternt (Georg) og internt (det øger spillets værdi for kunden).
- Prioritet: Senere implementation.
- Subkrav: 8
- Superkrav: 1

8: Mulighed for at spille med tal eller symboler

- Beskrivelse: Der skal være mulighed for at spille med symboler som alternativ til tal på 4x4 sudoku-pladerne.
- Eksempel: Brugeren kan vælge, om der skal spilles med tal eller foruddefinerede symbolsæt (sol, måne, stjerne, sky).
- Begrundelse: Dette krav vil gøre det nemmere for de mindste børn at forholde sig til spillet uden at kunne tallene.
- Kilder: Eksternt (spillerens progression, dvs. lærernes krav) og internt (det øger spillets værdi for kunden).
- Prioritet: Senere implementation.
- Subkrav: Ingen.
- Superkrav: 7

9: Feedback

- Beskrivelse: Spillet skal give løbende feedback på udførte handlinger og træk i form af farver og/eller lyd.
- Eksempel: Når der er udfyldt en korrekt række/søjle/kvadrat/sudoku, lyser det korrekt udfyldte op og der afspilles en fanfare.
- Begrundelse: Belønninger i løbet af spillet er med til at holde spillelysten oppe og skaber en følelse af, at barnet kommer videre og bliver belønnet.
- Kilder: Eksternt (Georg).
- Prioritet: Senere implementation.
- Subkrav: Ingen.
- Superkrav: 2

A.2. Baseline Design

Baseline Design

Dette dokument beskriver opbygningen af de enkelte lag i sudokuspillet. Mange overvejelser og store dele bygger på bogen ”Programming Sudoku” af Wei-Meng Lee. Heri er beskrevet algoritmer og kildekode til generering af symmetriske sudokuplader i flere sværhedsgrader. Til vores baseline design har vi valgt de dele af bogens implementation, der var nødvendige for at opfylde vores baselinekrav. Vi har valgt at bruge bogens kode som grundlag for vores model og en stor del af den fremtidige kodefase vil derfor bestå af oversættelse af koden til Java. I det følgende vil vi beskrive designet i 3 lag: overordnet arkitektur, modulopbygning og algoritmer og datastrukturer. Abstraktionsniveauet falder tilsvarende ned gennem de 3 lag.

Overordnet arkitektur

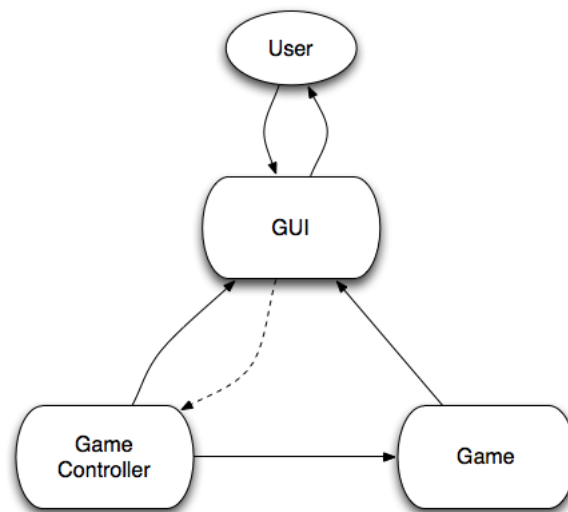
Som overordnet arkitektur har vi valgt at opdele spillet i 3 hovedkomponenter: grafisk brugerflade (GUI), Game Controller og Game. Motivationen for denne opdeling er flersidet:

- Ud fra vores oprindelige valg af Java som programmeringssprog ligger der væsentlige strukturelle fordele i denne opdeling. Opdelingen medfører, at vi kan anvende objektorienteret programmering i Java ved hjælp af Model-View-Control mønsteret (MVC). MVC harmonerer rigtig godt med vores opdeling, der altså giver os en overordnet fremgangsmåde for at opdele og strukturere vores moduler og kode i det hele taget.
- Opdelingen gør det ligeledes nemmere for gruppen at videreudvikle på en væsentlig del af projektet, nemlig den grafiske brugerflade. MVC-mønsteret gør det overskueligt for os at opbygge et baseline design, der virker, og herefter kunne videreudvikle på GUI'en uden at skulle ændre synderligt i Gamet.

Vi beskriver nu overordnet indholdet af de 3 komponenter, som kan ses skematisk i Figur 6:

GUI:

Her ligger alle elementerne i den grafiske brugerflade, dvs. de ting, som spilleren rent faktisk kan se og interagere med. Dette vil i vores tilfælde sige selve vinduet, indeholdende en spilleplade (af enten 9x9 eller 4x4 størrelse), mulighed for at vælge de tal, der skal indsættes (knapper med tallene fra enten 1-4 eller 1-9), mulighed for at starte et nyt spil (knap) og mulighed for at afslutte spillet (knap). GUI'et interagerer kun direkte med Game Controlleren, ikke med Gamet.



Figur 6: Skitse af overordnet arkitektur i baseline designet

Game Controller:

Denne komponent styrer samspillet mellem de to andre komponenter, idet den sørger for at modtage input fra GUI-komponenten, afgøre hvilken handling, der skal udføres på inputtet, og bede Game-komponenten om at udføre de spilhandlinger, der er nødvendige for at udføre det af brugeren ønskede træk eller programhandling.

Game:

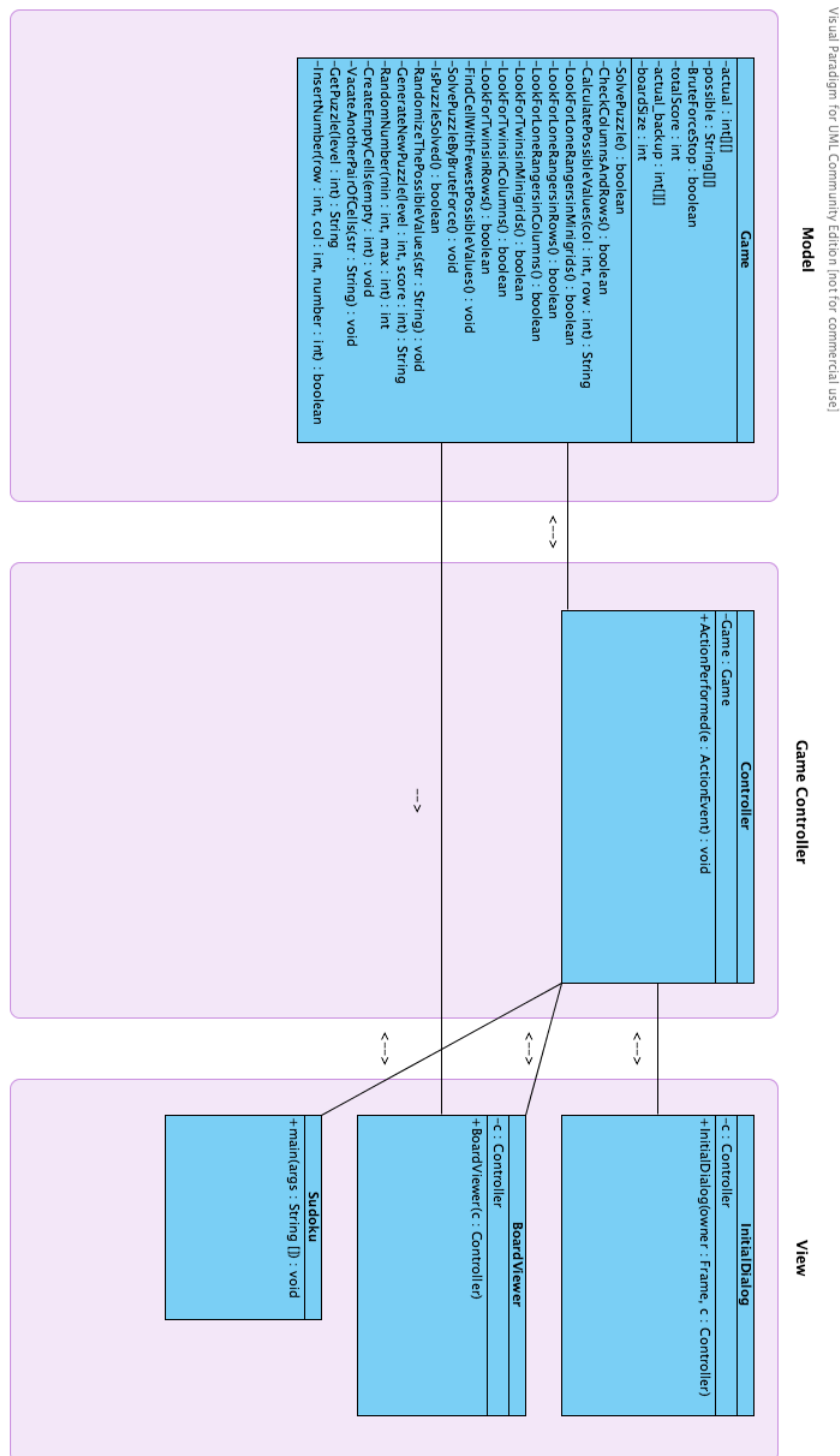
Her ligger selve spillet internt i programmet. Det er ligeledes kun denne komponent, der har lov til at udføre ændringer på pladen, ligesom det er også er her, at der kan genereres nye plader og afgøre, om sudokuen er løst. Altså indeholder modulet information om spillets status, dets videre forløb osv.

Moduldesign

I dette afsnit følger en dybere beskrivelse af de tre moduler gennemgået i ovenstående afsnit. Rent implementeringsmæssigt kodes alle moduler i Java 1.5 for at sikre bredest mulig kompatibilitet med målgruppens hardware og operativsystemer. I Figur 7 er vist et klassediagram over de tre moduler, indeholdende et udvalg af de vigtigste metoder og variable.

Model

Modellen indeholder i baseline kun én enkelt klasse, nemlig Game-klassen. Denne klasse indeholder selve det sudokuspil, der ligger "bag facaden", så at sige. Her ligger selve spillepladen (4x4 eller 9x9), her genereres nye plader, her udføres de træk, brugeren beder om (gennem Control-modulet), og her tjekkes, om et spil er færdigt. Klassen Game



Figur 7: Skitse af overordnet klassestruktur i baseline designet

indeholder et væsentligt antal mindre metoder til at udføre disse handlinger men er derudover ikke umiddelbart afhængig af de andre moduler. De enkelte metoder beskrives nærmere i næste afsnit. Udover nedennævnte *private* metoder findes et antal *public re-*

turnmetoder, der returnerer klassens primære metoders output, så disse metoder ikke kan tilgås udefra. I forhold til bogens implementation vil vi forsøge at generalisere metoderne, så de kan generere $n \times n$ -sudokuplader, hvor n er enten 4 eller 9; med mulighed for senere udvidelse til endnu større plader.

Class Game:

Vi gengiver nu kort de vigtigste metoder i Game-klassen. En nærmere implementeringsbeskrivelse af de vigtigste af disse kan findes i næste afsnit.

- **SolvePuzzle()**: Denne metode forsøger at løse sudokupladen kun ved brug af traditionelle metoder, dvs. uden at skulle gætte. Metoden gør brug af et væsentligt antal af klassens andre metoder (for at finde lone rangers, twins og lignende). Metoden er brugbar til processerne med at generere ny sudokuplader samt til at afgøre sværhedsgraden af de genererede plader.
- **SolvePuzzleByBruteForce()**: Denne metode forsøger at løse sudokupladen ved den gammeldags brute force-metode, hvor samtlige muligheder afprøves i rækkefølge. Metoden gør brug af flere hjælpemetoder i klassen og er en vigtig del af at generere en ny plade.
- **IsPuzzleSolved()**: Tjekker, om sudokupladen i dens nuværende udseende er løst.
- **InsertNumber()**: Forsøger at indsætte et tal på en bestemt plads på sudokupladen. Metoden skal dermed også afgøre, om brugeren overhovedet *må* indsætte et tal på det pågældende felt.
- **GetPuzzle()**: Sørger for at generere en sudokuplade med den rette sværhedsgrad.

Game Controller

Dette modul indeholder i baseline kun klassen Controller, der fungerer som bindeleddet mellem Game-modulet og View-modulet. Controller-klassen sørger for selve afviklingen af spillet, idet den afventer input fra View-modulets klasser (dvs. fra brugeren), hvorefter den igangsætter de påkrævede handlinger i Game-modulet (f.eks. starter et let spil, udfører et træk fra brugeren eller får Game-modulet til at svare på, om spillet er slut). Controlleren er afhængig af, at der for det første findes et View- og Game-modul (som kendes af Controlleren dels ved instantiering og dels ved at den selv opretter), ligesom den har brug for return-metoder i Game-klassen, så den kan udtrække information.

Class Controller:

Klassen Controller implementerer i Java en ActionListener, der lytter på View-modulets klasser (altså f.eks. brugerinterfacets knapper og sudokupladens felter) og sørger blandt andet for at oprette en instans af Game. Hovedmetoden er ActionPerformed(), der udfører handlinger baseret på brugerens input og klik.

View

Dette modul indeholder GUI'en, som i baseline designet er fordelt på 3 klasser: InitialDialog, BoardViewer og Sudoku. Alle klasser i View-modulet er afhængige af Java-udvidelserne `javax.swing.*`, der indeholder de grafiske moduler, som brugerfladen skal opbygges af. Udformningen i Java gør desuden, at brugergrænsefladen bør se næsten ens ud på forskellige platforme. De beskrives kort herunder:

Class InitialDialog:

Denne klasse er den grafiske brugerflade til opstartsdialogen, hvor brugeren vælger sværhedsgraden af spillet (4x4 eller 9x9). De to muligheder tilføjes `ActionListeners`, som giver Controlleren besked om, hvad der er valgt i dialogen. Interfacedesignet forventes at blive lavet i et `GridBagLayout`, ligesom det ikke er nødvendigt med en menu (jævnfør senere overvejelser i forhold til GUI'en).

Class BoardViewer:

BoardViewer indeholder den grafiske brugerflade til selve spillet, hvor sudokupladen vises og interageres med, ligesom der forefindes knapper til navigation (Nyt spil, Slut, Hjælp) og udførsel af træk (knapper med tallene 1-9). Designet forventes at blive lavet i et `GridBagLayout`, ligesom det ikke er nødvendigt med en menu (jævnfør senere overvejelser i forhold til GUI'en).

Class Sudoku:

Sudoku-klassen er "hovedklassen", idet den har `main`-metoden liggende, der sørger for at starte de resterende moduler op, når programmet startes. Klassen instansierer en Controller, en BoardViewer og en InitialDialog, hvorefter Controlleren parres med de to andre via `Observers`, og kontrollen overgives til Controlleren.

Algoritmer og datastrukturer

Vores sudokuplader har vi valgt at repræsentere som todimensionelle arrays af heltal (`int[][]`). Der er her tale om `actual` og `actual_backup`. Desuden anvendes et todimensionelt array af Strings til at holde styr på de mulige værdier for hvert felt på pladen (`String[][]`). Grunden til disse valg er til dels, at de følger bogens implementation, samt at datastrukturerne er forholdsvis simple. Disse valg var dog allerede besluttet på forhånd.

Herunder præsenteres nu pseudokode for algoritmen til generering af sudokuplader (forefindes i Game-klassen):

```
GetPuzzle(integer minEmptyCells, integer maxEmptyCells)
{
    return GenerateNewPuzzle(minEmptyCells, maxEmptyCells)
}
```

```

GenerateNewPuzzle( integer minEmptyCells, integer maxEmptyCells)
{
Initialize puzzle by creating a double array with integers (zeros)
in each cell and a double array with the string containing all the
possible values in the cell.

if not SolvePuzzle() (If the puzzle cannot be solved by logical methods)
then SolvePuzzleByBruteForce

CreateEmptyCells(RandomNumber(minEmptyCells,maxEmptyCells))
(Create a randomized number of empty cells in pairs so the sudoku
is symmetrical)

int tries = 0
while not SolvePuzzle() and tries < 50
    VacateAnotherPairOfEmptyCells(the array as a string)
    (i.e. fill in a pair of cells and vacate two others)
    integer tries = tries + 1
if not SolvePuzzle GenerateNewPuzzle(minEmptyCells, maxEmptyCells)

return the puzzle
}

SolvePuzzleByBruteForce()
{
Determine the cell with fewest possible values via
the method FindCellWithFewestPossibleValues.
Determine the possible values for the cell and radomize them.
i = 0
while the sudoku cannot be solved and i < the number of
possible values in the cell.
    Put the i'th possible value in the cell and check if
the sudoku is solvable via the method SolvePuzzle.
    if the sudoku is not solved
        Try to fill in another cell by running SolvePuzzleByBruteForce again
}

FindCellWithFewestPossibleValues(c, r)
{
min = 10
for each cell if the number of possible values < min
    then min = the number of possible values of the cell
    save the column number and row number of the cell in c and r
}

SolvePuzzle()
{
while the sudoku still changes and is not solved
    do look for twins in rows, columns and minigrids
    while the sudoku still changes and is not solved
        do look for lone rangers rows, columns and minigrids

```

```

        while the sudoku still changes and is not solved
            do perform CRME
if IsPuzzleSolved()
    then return true
    else return false
}

Boolean IsPuzzleSolved()
{
For each row check if the numbers from 1 to 9 are there.
For each column check if the numbers from 1 to 9 are there.
For each minigrid check if the numbers from 1 to 9 are there.
}

CalculatePossibleValues() (calculates possible values for a cell)
{
Remove from possible values, all actual values of cells in the same row
                                -||-                column
                                -||-                minigrid

}

CheckColumnsAndRows() (calculates possible values for all cells)
{
calls CalculatePossibleValues() for each cell
if amount of possible values = 1
    insert number
}

LookForLoneRangersinRows()
{
for each row
    count for each number
    amount of possibilities

    If possibilities = 1
        Confirm number
        Insert number
}

LookForLoneRangersinColumns()
{
for each column
    count for each number
    amount of possibilities

    If possibilities = 1
        Confirm number
        Insert number
}

LookForLoneRangersinMinigrids()

```

```
{
for each minigrid
    count for each number
    amount of possibilities

    If possibilities = 1
        Confirm number
        Insert number
}

LookForTwinsinRows()
{
for each row
    count for each cell
    amount of possibilities

    if amount of possibilities = 2 ( possibilities = "xy")
        compare possible values to rest of cells in row

        if equality found
            remove x from possible values of the cells in the row except the twins
            remove y from possible values of the cells in the row except the twins

            if new amount of possibilities = 0
                throw exception

            if new amount of possibilities = 1
                insert number
}

LookForTwinsInColumns()
{
For each column
    count for each cell
    amount of possibilities

    if amount of possibilities = 2 ( possibilities = "xy")
        compare possible values to rest of cells in column

        if equality found
            remove x from possible values of the cells in the column except the twins
            remove y from possible values of the cells in the column except the twins

            if new amount of possibilities = 0
                throw exception

            if new amount of possibilities = 1
                insert number
}

LookForTwinsinMinigrids()
{
for each minigrid
```

```

count for each cell
amount of possibilities

if amount of possibilities = 2 ( possibilities = "xy")
    compare possible values to rest of cell in minigrid

    if equality found (the cell has a twin)
        remove x from possible values of the cells in the minigrid except the twins
        remove y from possible values of the cells in the minigrid except the twins

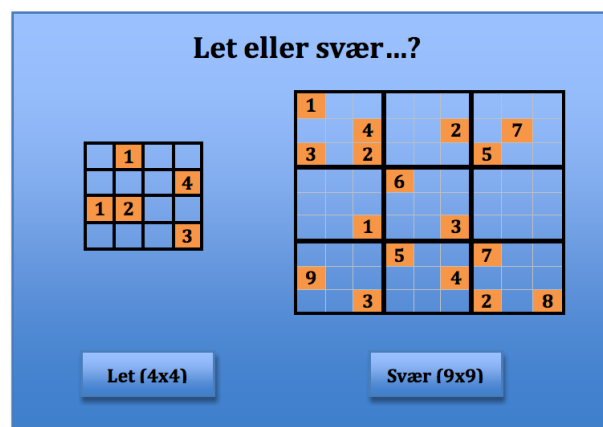
        if new amount of possibilities = 0
            throw exception

        if new amount of possibilities = 1
            insert number
}

```

Overvejelser omkring GUI

I Figur 8 ses en skitse af, hvordan vi på nuværende tidspunkt kunne forestille os, at klassen InitialDialog vil komme til at se ud. Denne dialog ses som det første, når spillet startes, og når brugeren vælger at starte et nyt spil. Vægten er lagt på klare farver og simpelt design, med kun to knapper: én til at starte et let spil (4x4) og én til at starte et sværere spil (9x9).



Figur 8: Skitse af InitialDialog-klassen

Brugeren vil derefter blive taget til selve spillepladen (se Figur 9), her vist for en 9x9-plade. Igen er vægten lagt på klare farver og simpelt design, med så lidt tekst og knapper som muligt. Brugersiden består her af en knap til at starte et nyt spil, en knap til at slutte programmet, en vink-knap til at give hints undervejs i spillet, selve sudokupladen og til sidst 9 (hhv. 4) "bunker tal" i bunden, som viser de forskellige tal, der kan sættes ind på pladen. Når der ikke er flere af et bestemt tal, vises "bunken" i en anden farve, eksempelvis grå. Selve spillet foregår så ved, at brugeren klikker på en af bunkerne og klikker derefter på et tomt felt på pladen for at placere tallet.

Hint-knappen er medtaget, selvom den ikke umiddelbart er i baseline, da den vil være første prioritet til implementering udover baseline. Desuden kan der laves forbedringer i knapperne på følgende måder: alle knapper får deres eget sigende symbol ved siden af teksten (genkendelighed fra andre typer spil, som børnene har prøvet), samt at knapperne får hver deres farve. Dette vil gøre kommunikationen mellem en eventuel underviser/forælder og barnet lettere, da farver er lettere for små børn at relatere til end tekst ("tryk på den røde knap" i stedet for "tryk på knappen med Slut").



Figur 9: Skitse af BoardViewer-klassen

Opfyldelse af kravspecifikation

Krav nummer 1 om generering af valide sudokuer er til dels opfyldt, da vi mangler en overvejelse af, hvordan en 4x4 skal genereres. 9x9 er dog opfyldt med algoritmen i bogen.

Krav nummer 2 om simpel grafisk brugerflade vil være opfyldt, hvis ovenstående overvejelser om design af GUI følges til dørs i det endelige design. Det vil dog ikke som udgangspunkt være opfyldt i baseline.

Krav nummer 3 om lav responstid bør kunne opfyldes med den valgte implementation

og algoritme. Dette vil dog ikke være muligt at teste, før vi ligger inde med en baseline-kode.

Krav nummer 6 om nem installation og afvikling vil kunne opfyldes, idet MVC-mønsteret muliggør sammenpakning af programmet i én fil, der er platformsuafhængig og kan startes med et dobbeltklik, forudsat at der er installeret mindst Java 1.5 på brugerens computer.

Altså vil baseline-kravene alle kunne opfyldes med nærværende valg af design og implementation. For resten af kravenes vedkommende vil nogle være delvist opfyldt, andre slet ikke. Dette vælger vi dog at se bort fra i denne gennemgang.

Overvejelser omkring valg af design

I de næste to afsnit beskrives fordele og ulemper ved vores valg af designstrategier, modellering og modularisering.

Fordele ved designet

Vi opsummerer her fordelene ved de valg, der er truffet mht. overordnet design, designmønster og datastrukturer.

- MVC er efter vores mening det mest oplagte designmønster til denne opgave, da programmets struktur og simple opbygning lægger op til en tredeling af modulerne.
- MVC gør det nemt senere at udbygge med flere features og designændringer, da Model-komponenten dybest set er uafhængig af Game Controller og View. Dvs. at så snart modellen virker og kan levere de påkrævede data, kan denne ses som færdig, og der kan f.eks. udvikles videre på GUI'en.

Herudover skal det nævnes, at MVC i vores design og implementation medfører en høj cohesion og lav coupling. Grunden til det første er, at der specielt i Game-klassen intensivt bliver brugt mange små metoder, hvor størstedelen har højst to argumenter og alle udfører hver deres specifikke opgave. Dette må tilskrives bogens forholdsvist gode modulering af koden i metoder/funktioner. Dog er der eksempler på både temporal cohesion (i constructor-metoderne) og communicational cohesion, men mestendels er der tale om functional cohesion, som er at foretrække.

Den lave coupling er et resultat af, at f.eks. Game-klassen kun tillader dens egne private metoder at ændre i datastrukturer i klassen, ligesom data, der går ud af klassen, håndteres af klassens egne return-metoder. Ligeledes muliggør MVC en naturlig adskillelse af data, og langt de fleste metodekald mellem klasserne foregår vha. et lille antal

(eller ingen) rene dataparametre.

Den modulære opdeling i MVC muliggør en forholdsvis simpel testplan, da man vil kunne teste metoder for sig og moduler for sig. Størstedelen af testarbejdet vil dog for vores implementations vedkommende ligge i Game-klassen, hvor alt det ”interne” arbejde foregår. Test af View er svært at automatisere, udover at kontrollere, at brugerfladen reagerer som planlagt. Det samme gælder for test af Game Controller-modulet.

Nærværende beskrivelse af designet er baseret på *breadth first*-fremgangsmetoden, som giver et højt niveau af abstraktion; altså kan man selv på højniveau-beskrivelsen danne sig et indtryk af modulernes funktionalitet uden at kende f.eks. hvilke datastrukturer og algoritmer, de enkelte metoder anvender.

Ulemper ved designet

Der er visse ulemper ved vores valg af design. I vores Game-klasse er der et forholdsvis stort antal metoder, hvilket kan give problemer med et danne sig et overblik over, hvilke metoder der laver hvad. Her kan man på et senere tidspunkt vælge at dele Game-klassen op i mindre underklasser med relaterede metoder og funktioner.

Andre ulemper vil sandsynligvis dukke op under kode- og testfasen, men på nuværende tidspunkt er det for svært at opdage flere.

Konklusion

Ovenstående design og skitse af implementation lægger efter vores mening en god bund for den fremtidige udvikling af spillet. Specielt kodefase vil være relativt simpel at gå i gang med nu ud fra vores pseudokode og det faktum, at meget af kodningen består i oversættelse. Ligeledes har vi fået beskrevet sammenhængen mellem modulerne på en sådan måde, at implementeringen af dem kan foregå så separat som muligt. Alt i alt et lovende design.

A.3. Endeligt Design

Endeligt Design

Dette dokument beskriver opbygningen af de enkelte lag i sudokuspillet. Mange overvejelser og store dele bygger på bogen ”Programming Sudoku” af Wei-Meng Lee. Heri er beskrevet algoritmer og kildekode til generering af symmetriske sudokuplader i flere sværhedsgrader. Til vores design har vi valgt de dele af bogens implementation, der var nødvendige for at opfylde vores krav. Vi har oversat den kode fra bogen, som vi skulle bruge, til Java og modificeret den i henhold til vores behov og anderledes datastrukturer. Mere herom senere. I det følgende vil vi beskrive designet i 3 lag: overordnet arkitektur, modulopbygning og algoritmer og datastrukturer. Abstraktionsniveauet falder tilsvarende ned gennem de 3 lag.

Overordnet arkitektur

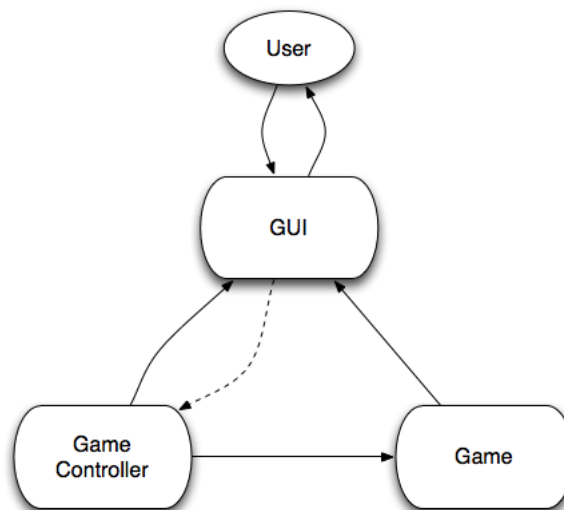
Som overordnet arkitektur har vi valgt at opdele spillet i 3 hovedkomponenter: grafisk brugerflade (GUI), Game Controller og Game. Motivationen for denne opdeling er følgende:

- Ud fra vores oprindelige valg af Java som programmeringssprog ligger der væsentlige strukturelle fordele i denne opdeling. Opdelingen medfører, at vi kan anvende objektorienteret programmering i Java ved hjælp af Model-View-Control mønsteret (MVC). MVC harmonerer rigtig godt med vores opdeling, der altså giver os en overordnet fremgangsmåde for at opdele og strukturere vores moduler og kode i det hele taget.
- Opdelingen gør det ligeledes nemmere for gruppen at videreudvikle på en væsentlig del af projektet, nemlig den grafiske brugerflade. MVC-mønsteret gør det overskueligt for os at opbygge et design, der virker, og herefter kunne videreudvikle på GUI'en uden at skulle ændre synderligt i Gamet.

Vi beskriver nu overordnet indholdet af de 3 komponenter, som kan ses skematisk i Figur 10:

GUI:

Her ligger alle elementerne i den grafiske brugerflade, dvs. de ting, som spilleren rent faktisk kan se og interagere med. Dette vil i vores tilfælde sige selve vinduet, indeholdende en spilleplade (af enten 9x9 eller 4x4 størrelse), mulighed for at vælge de tal, der skal indsættes (knapper med tallene fra enten 1-4 eller 1-9), mulighed for at starte et nyt spil (knap) og mulighed for at afslutte spillet (knap). GUI'et interagerer direkte med Game Controlleren og henter data fra Gamet.



Figur 10: Skitse af overordnet arkitektur i designet

Game Controller:

Denne komponent styrer samspillet mellem de to andre komponenter, idet den sørger for at modtage input fra GUI-komponenten, afgøre hvilken handling, der skal udføres på inputtet, og bede Game-komponenten om at udføre de spilhandlinger, der er nødvendige for at udføre det af brugeren ønskede træk eller programhandling. Game Controlleren sørger ligeledes for at starte selve spillet ved at instantiere de rigtige klasser og lave forbindelser mellem dem.

Game:

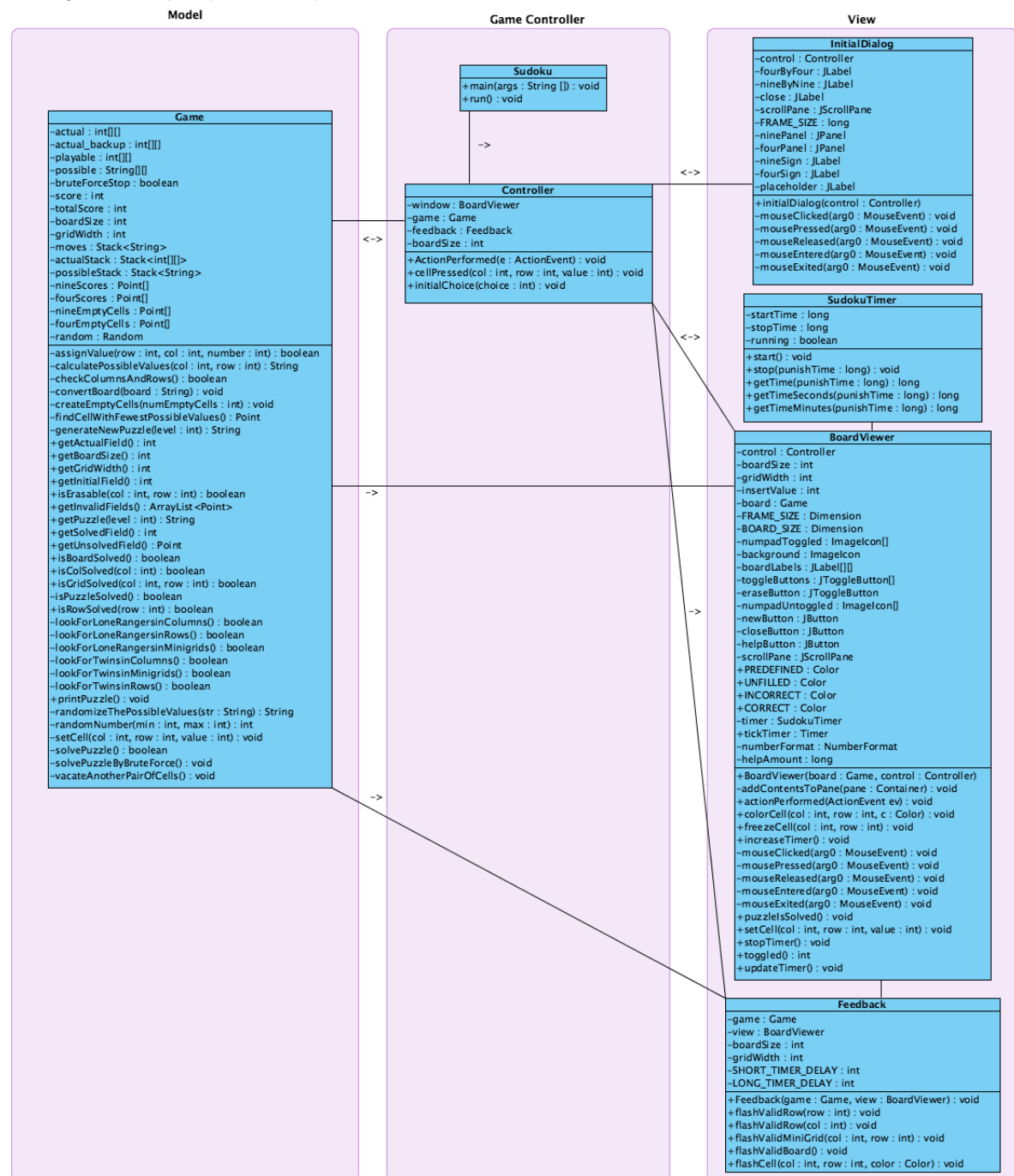
Her ligger selve spillet internt i programmet. Det er ligeledes kun denne komponent, der er i stand til at udføre konkrete ændringer på pladen, ligesom det er også her, at der kan genereres nye plader og afgøres, om sudokuen er løst. Altså indeholder modulet information om spillets status, dets videre forløb osv.

Moduldesign

I dette afsnit følger en dybere beskrivelse af de tre moduler gennemgået i ovenstående afsnit. Rent implementeringsmæssigt kodes alle moduler i Java 1.5 for at sikre bredest mulig kompatibilitet med målgruppens hardware og operativsystemer. I Figur 11 er vist et klassediagram over de tre moduler, indeholdende et udvalg af de vigtigste metoder og variable.

Model**Class Game:**

Denne klasse er ansvarlig for at konstruere en sudoku, samt at kunne udføre træk



Figur 11: Skitse af overordnet klassestruktur i designet

på pladen (at indsætte tal m.m.). Det vil sige, at den hele tiden skal vide, hvordan det aktuelle spil ser ud efter hvert træk, samtidig med at den skal vide, hvordan pladen kan løses. For at kunne videregive denne information indeholder den en række return-metoder som GUI'et og Controlleren henter information om spillet

fra. Desuden indeholder den nogle metoder, som Controlleren kan benytte til at fortælle klassen om træk, der er sket på spillepladen i GUI'et.

Game Controller

Class Controller:

Controller-klassen håndterer interaktionen mellem de andre klasser. Den er ansvarlig for at modtage information fra GUI'et om brugerens handlinger i spillet og efterfølgende udføre disse handlinger i modellen eller GUI'et. Klassen modtager input fra GUI'et, forespørger modellen, ændrer i modellen og giver besked til GUI'et igen om eventuelle opdateringer. Ligeledes beder Controlleren brættet om at give feedback til brugeren og sørger for at oprette nye spil, håndtere hjælpesystemet og timeren og afslutte programmet.

Class Sudoku:

Sudoku-klassen er "hovedklassen", idet den har main-metoden liggende, der sørger for at starte de resterende moduler op, når programmet startes. Klassen instantierer en Controller, og kontrollen overgives til denne. Dette er en ændring i forhold til baseline designet, hvor Sudoku-klassen også stod for at instantiere en BoardViewer og en InitialDialog. Sudoku-klassen er den eneste i programmet indeholdende en main-metode.

View

Dette modul indeholder GUI'et, som i designet er fordelt på 3 klasser: InitialDialog, BoardViewer og Feedback. Alle klasser i View-modulet er afhængige af Java-udvidelserne javax.swing og java.awt, der indeholder de grafiske moduler, som brugerfladen skal opbygges af. Udformningen i Java gør desuden, at brugergrænsefladen bør se næsten ens ud på forskellige platforme. De beskrives kort herunder:

Class InitialDialog:

Denne klasse er den grafiske brugerflade til opstartsdialogen, hvor brugeren vælger sværhedsgraden af spillet (4x4 eller 9x9). De to muligheder tilføjes MouseListeners, som giver Controlleren besked om, hvad der er klikket på i dialogen. Herefter kommer man videre til BoardVieweren. Interfacedesignet er lavet i et GridBagLayout, og det er ikke nødvendigt med en menu (jævnfør senere overvejelser i forhold til GUI'et [A.3](#)).

Class BoardViewer:

BoardViewer indeholder den grafiske brugerflade til selve spillet, hvor sudokupladen vises og interageres med, ligesom der forefindes knapper til navigation (Nyt spil, Slut, Hjælp) og udførsel af træk (knapper med tallene 1-4 eller 1-9). Den viser også den timer, der er pointsystemet i spillet. Designet er lavet i et GridBagLayout

ved brug af diverse elementer fra javax.swing pakken.

Klassen er ansvarlig for al interaktionen med brugeren, når spillet spilles. Den er afhængig af klassen Game i den forstand, at den henter information om det aktuelle spil herfra. Den er afhængig af Feedback-klassen og SudokuTimeren til dele af GUI'et hvor disse elementer benyttes. Desuden er den afhængig af at Controlleren lytter til handlinger og tager sig af dem. Dette sker ved at et klik på en knap aktiverer enten en ActionEvent, som Controlleren tager sig af, ellers kalder en MouseEvent metoder fra Controlleren. BoardViewer er også selv ActionListener på handlingerne i GUI'et, så den kan afgøre, om der er valgt et tal, der skal indsættes, når der klikkes på et tomt felt eller om der ikke skal ske noget. Se i øvrigt overvejelser i forhold til GUI'et A.3.

Class Feedback:

Feedback indeholder metoder til at give feedback til spilleren. Klassens hovedansvar er dermed at give feedback, når viewet forespørger om det. Når der i viewet indsættes et tal, forespørger Controlleren modellen om en række/søjle/minigridd/sudoku er løst, og hvis det er tilfældet kommer forskelligt visuelt feedback. Den nærmere beskrivelse af den visuelle feedback kan ses i afsnittet om overvejelser i forhold til GUI'et A.3.

Class SudokuTimer:

SudokuTimer er ansvarlig for at skabe en ny timer, der kan benyttes i sudokuen. Det vil sige den skal have metoder, der returnerer den tid, som er gået fra timeren blev sat i gang, og som samtidig tager højde for om den stadig kører eller ej. Vi har valgt at implementere denne ud fra eksemplet på <http://www.goldb.org/stopwatchjava.html>. Desuden er der en return metoder der angiver både millisekunder, sekunder og minutter, der er gået.

Algoritmer og datastrukturer

Vores sudokuplader har vi valgt at repræsentere som todimensionelle arrays af heltal (`int[][]`). Der er her tale om `actual` og `actual_backup`. Desuden anvendes et todimensionelt array af Strings til at holde styr på de mulige værdier for hvert felt på pladen (`String[][]`). Grunden til disse valg er til dels, at de følger bogens implementation, samt at datastrukturerne er forholdsvis simple. Disse valg var dog allerede besluttet på forhånd. I nogle tilfælde bliver sudokuerne og dele heraf dog også repræsenteret af strenge, fordi vi ønsker at holde os så tæt op ad "Programming Sudoku" som muligt, indtil vi er sikre på at alting virker som det skal. Vi er ikke klar over hvorfor han har valgt at gøre det således idet det så ikke er muligt at lave sudokuer større end 9x9 siden der i dem kan forekomme tocifrede tal.

Den eneste meget vigtige algoritme, der ikke er let gennemskuelig er vores algoritme i modellen til at generere sudokuplader. Der har været forskellige af disse på banen som det ses af vores mødereferater, men vi endte med at benytte den fra "Programming Sudo-

ku” med få variationer. Udførlig pseudokode for denne ses i Baseline Design afleveringen og vil ikke blive medtaget her. I stedet følgende noget mere kortfattede beskrivelse:

En sudokuplade udfyldes på følgende måde:

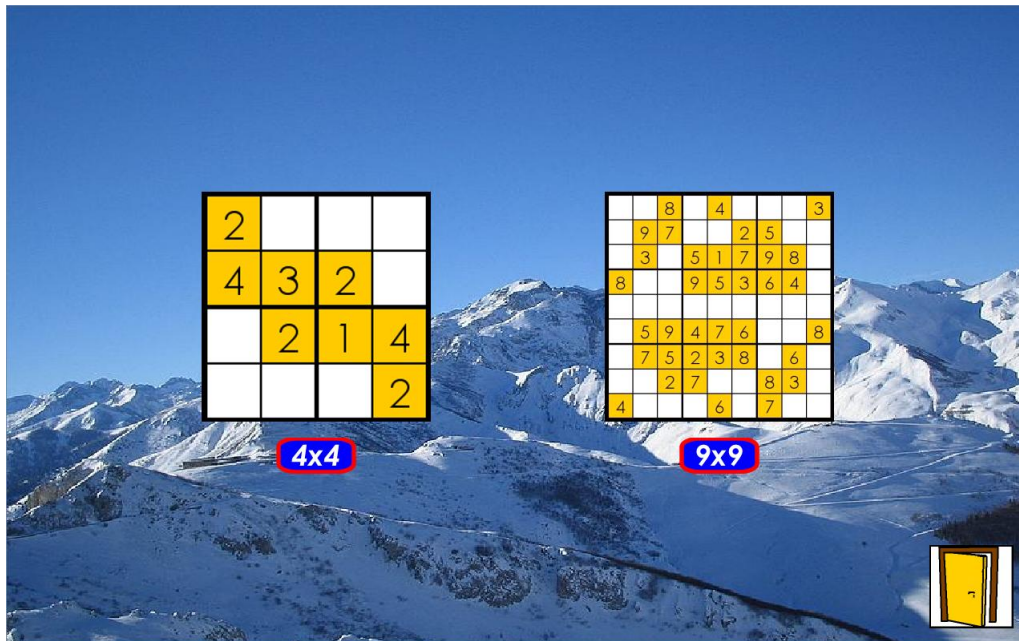
1. De mulige værdier i hvert felt angives ved ad omveje at køre `calculatePossibleValues`.
2. `solvePuzzleByBruteForce`-metoden udfylder hele sudokupladen ved at køre følgende loop
 - udvælger et felt med færrest mulige værdier,
 - sætter et tal ind,
 - indsætter om muligt tal ud fra logiske regler og opdaterer possible værdierne via `solvePuzzle`-metoden.
3. Hvis dette giver en udfyldt plade returneres denne, ellers begyndes forfra.
4. Vha. `createEmptyCells`-metoden skabes tomme felter ud fra det angivne niveau.
5. Hvis ikke sudokuen kan løses benyttes `vacateAnotherPairOfCells`-metoden indtil den kan løses eller indtil der er forsøgt så mange gange, at den begynder forfra med at generere en ny sudoku.

Fordelene ved denne metode er, at der ingen backtracking er og idet der benyttes logiske regler til at finde værdier er der stor sandsynlighed for succes. Ulempen er, at det kan tage flere forsøg at skabe en helt udfyldt plade. Der er også her vores kode specielt adskiller sig fra den i ”Programming Sudoku”; i bogens kode er der backtracking, hvis der kastes exceptions. Vi mener ikke, det er en god idé på den måde at udelukke at exceptions kan kastes af andet end en sudoku der skal backtracks, så det har vi ikke gjort. Desuden virker datastrukturen (stakke) fra bogen ikke til vores dobbeltarrays, så vi har heller ikke en ordentlig måde at backtracke på. Derudover har vi set, at det kun er meget få gange, det er nødvendigt at lave en helt ny sudoku og på denne måde er det acceptabelt at undlade at backtracke.

Overvejelser omkring GUI

I Figur 12 ses vores nuværende udgave af klassen `InitialDialog`. Denne dialog ses i fuld skærm som det første, når spillet startes, og når brugeren vælger at starte et nyt spil. Vægten er lagt på klare farver og simpelt design, med kun tre knapper: én til at starte et let spil (4x4), én til at starte et sværere spil (9x9) og en knap til at afslutte programmet (døren).

Brugeren vil derefter blive taget til selve spillepladen (se Figur 13), her vist for en 9x9-plade, der også er i fuld skærm. Igen er vægten lagt på klare farver og simpelt design, med så lidt tekst og så få knapper som muligt. Brugersiden består her af



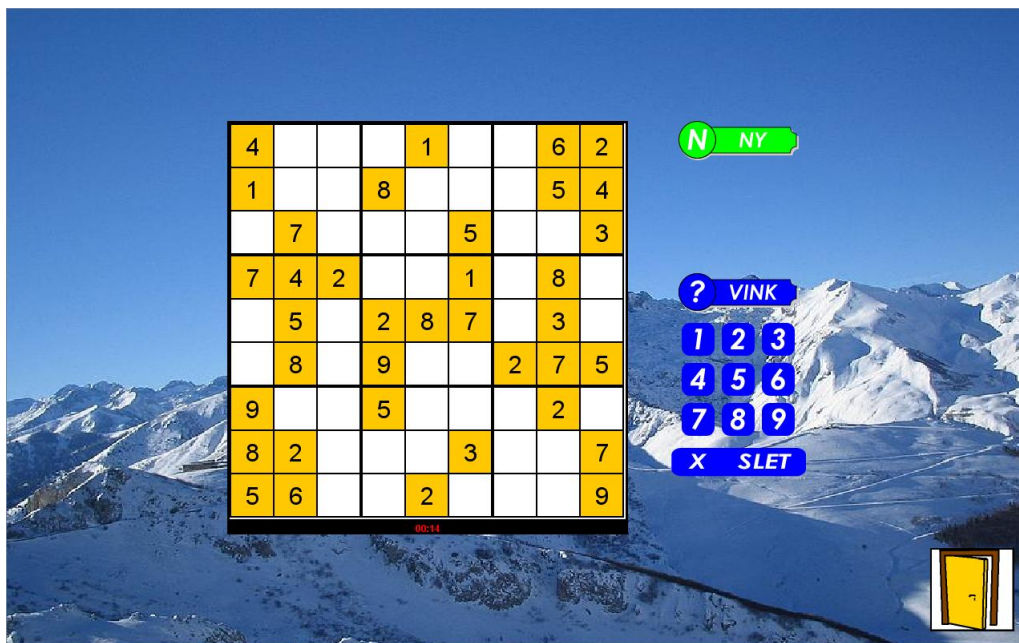
Figur 12: Skitse af InitialDialog-klassen

- en knap til at starte et nyt spil (en ny InitialDialog),
- en knap til at slutte programmet,
- en vink-knap til at give hints undervejs i spillet,
- selve sudokupladen,
- et telefontastatur med 9 (hhv. 4) knapper, som viser de forskellige tal, der kan sættes ind på pladen,
- en slet-knap, så et tal kan slettes hvis det er forkert, samt
- en timer under spillepladen, som er vores bud på et pointsystem, der kan få brugeren til at øve sig og dermed blive bedre, samt at få flere brugere til at konkurrere.

Timeren begynder så snart billedet kommer frem og stopper, når pladen er korrekt udfyldt. Selve spillet foregår så ved, at brugeren klikker på et af tallene og derefter klikker på et tomt felt på pladen for at placere tallet. Der kan ikke indsættes tal i de gule felter, hvor der allerede står tal. Hvis det indsatte tal færdiggør en række/søjle/minigridd blinker denne grøn og farves derefter gul ligesom de forudfyldte felter. På denne måde er det let at følge med i hvor meget man mangler. Hvis man ønsker at slette et forkert tal trykkes på Slet-knappen og så på det felt hvor man vil fjerne et tal. Dette kan kun lade sig gøre på de hvide felter. Når pladen er udfyldt korrekt, vises tastaturet samt knapperne vink og slet i en anden farve, grå, og minigridds blinker i forskellige farver indtil der trykkes på enten Ny- eller Luk-knappen.

Hvis man i løbet af spillet ikke kan komme videre, kan Hint-knappen aktiveres. Så markeres de forkert indsatte tal med rød, hvis der er nogen, og fjernes så. Hvis alle de indsatte tal er korrekte indsættes et nyt tal i stedet, og placeringen bliver i dette tilfælde markeret med et grønt blink. For at Hint-knappen ikke bruges skruppelløst, koster et hint et bestemt antal sekunder lagt til timeren (det bliver mellem 20 og 50 sekunder), så det ikke kan betale sig at hinte sig hele vejen igennem en sudoku.

Det bemærkes, at alle knapperne har enten en særlig farve eller et særligt symbol, som børnene kan genkende. Dette vil desuden gøre kommunikationen mellem en eventuel underviser/forælder og barnet lettere, da farver og symboler er lettere for små børn at relatere til end tekst ("tryk på døren" i stedet for "tryk på knappen med Slut"). Desuden er begge skærbilleder i fuld skærm, hvilket betyder, at der ikke er forstyrrende momenter fra menuer og andre elementer på computerskærmen, der kan forvirre og fjerne fokus fra spillet, når det køres.



Figur 13: Skitse af BoardViewer-klassen

Opfyldelse af kravspecifikation

Krav 1: *Algoritme til generering af valide sudokuer* Dette krav er opfyldt for pladestørrelserne 4x4 og 9x9. Her kan vores model generere entydige, symmetriske sudokuplader i en passende sværhedsgrad for børn i målgruppen.

Krav 2: *Simpel grafisk brugerflade* Dette krav er opfyldt, idet vores program har et minimalt antal knapper, en minimal mængde tekst og samtidig kører i fuld skærm (hvor det er muligt). Dette giver et mere roligt indtryk af spillet, da der ikke forefindes andre forstyrrende elementer fra f.eks. styresystemet undervejs.

- Krav 3:** *Lav responstid* Alle spillets handlinger (på nær opstart) kan udføres inden for 2 sekunder (ofte en del hurtigere), hvilket er testet på indtil videre 3 forskellige systemer (Mac og Windows). Dette inkluderer også generering af valide plader, hvormed kravet er opfyldt.
- Krav 4:** *Pointsystem* Dette krav er af tidsmæssige årsager kun delvist opfyldt. Der er blevet implementeret en timer på skærmen, der starter, når en ny plade er genereret og stopper, når pladen er løst. Desuden lægges et antal sekunder til, hver gang spilleren anvender hjælpesystemet.
- Krav 5:** *Hjælpesystem* Kravet er opfyldt med nærværende implementation. I denne fungerer systemet ved, at når der klikkes på hjælp, fjernes eventuelt forkert indsatte tal fra pladen. Er der ingen forkert indsatte tal, indsættes i stedet et nyt vilkårligt rigtigt tal på pladen. Systemet fungerer vha. en enkelt knap. Når denne knap aktiveres lægges et bestemt antal sekunder til timeren, så det ikke kan betale sig at trykke hjælp hver gang, der skal indsættes et tal.
- Krav 6:** *Nem installation og afvikling* Spillet kan distribueres som én fil (freeregnet brugervejledningen) og er udviklet i Java. Desuden er der valgt kompatibilitet med Java 1.5, da 1.6 endnu ikke er nok udbredt. Spillet kan desuden startes med et dobbeltklik af brugeren og gennemføres kun ved brug af en mus. Kravet er hermed opfyldt.
- Krav 7:** *Mulighed for flere sværhedsgrader* Dette krav er delvist implementeret, da brugeren har mulighed for at vælge mellem 4x4- eller 9x9-plader. Modellen i dens nuværende udseende understøtter desuden forskellige sværhedsgrader inden for disse to kategorier, men af tidsmæssige og pædagogiske årsager har vi valgt ikke at implementere det i GUI'et. Vi henviser til rapporten for nærmere begrundelse.
- Krav 8:** *Mulighed for at spille med tal eller symboler* Dette krav er blevet fraveget efter samtale med forskellige skolelærere. Vi henviser igen til rapporten for en nærmere begrundelse.
- Krav 9:** *Feedback* Når spilleren har udfyldt en korrekt række/søjle/minigridd, svarer programmet ved at markere den pågældende række/søjle/minigridd med grønt i et kort tidsrum. Derefter bliver rækken/søjlen/minigridd'et gult ligesom de felter der var indsat fra begyndelsen. Desuden blinker minigridds i forskellige farver indtil der klikkes på en knap, når sudokuen er løst.

Overvejelser omkring valg af design

I de næste to afsnit beskrives fordele og ulemper ved vores valg af designstrategier, modellering og modularisering.

Fordele ved designet

Vi opsummerer her fordelene ved de valg, der er truffet mht. overordnet design, designmønster og datastrukturer.

- MVC er efter vores mening det mest oplagte designmønster til denne opgave, da programmets struktur og simple opbygning lægger op til en tredeling af modulerne.
- MVC gør det nemt senere at udbygge med flere features og designændringer, da Model-komponenten dybest set er uafhængig af Game Controller og View. Dvs. at så snart modellen virker og kan levere de påkrævede data, kan denne ses som færdig, og der kan f.eks. udvikles videre på GUI'et.

Herudover skal det nævnes, at MVC i vores design og implementation medfører en høj cohesion og lav coupling. Grunden til det første er, at der specielt i Game-klassen intensivt bliver brugt mange små metoder, hvor størstedelen har højst to argumenter og alle udfører hver deres specifikke opgave. Dette må tilskrives bogens forholdsvis gode modulering af koden i metoder/funktioner. Dog er der eksempler på både temporal cohesion og communicational cohesion, men mestendels er der tale om functional cohesion, som er at foretrække.

Den lave coupling er et resultat af, at f.eks. Game-klassen kun tillader sine egne private metoder at ændre i datastrukturer i klassen, ligesom data, der går ud af klassen, håndteres af klassens egne return-metoder. Ligeledes muliggør MVC en naturlig adskillelse af data, og langt de fleste metodekald mellem klasserne foregår vha. et lille antal (eller ingen) rene dataparametre.

Nærværende beskrivelse af designet er baseret på *breadth first*-fremgangsmetoden, som giver et højt niveau af abstraktion; altså kan man selv på højniveau-beskrivelsen danne sig et indtryk af modulernes funktionalitet uden at kende f.eks. hvilke datastrukturer og algoritmer, de enkelte metoder anvender.

Ulemper ved designet

Der er visse ulemper ved vores valg af design. I vores Game-klasse er der et forholdsvis stort antal metoder, hvilket kan give problemer med at danne sig et overblik over, hvilke metoder der laver hvad. Det ville muligvis have været en fordel at dele Game-klassen op i mindre underklasser med relaterede metoder og funktioner.

Nogle af de andre klasser kunne også godt moduleres bedre, så der fx var en `TickTimer` klasse i stedet for at den er implementeret i `BoardViewer` eller at der var en hjælpeklasse i stedet for at hjælpen er implementeret i `Controller`, men pga. tidsnød har vi ikke gjort det. Det sidste eksempel ville også have gjort det nemmere at implementere andre former for hjælp end netop den vi har valgt.

Et andet problem er, at den kode vi har oversat fra "Programming Sudoku" kun håndterer 9x9 sudokuer. Så vi har været nødt til at lave om i en del af metoderne, for at det var muligt at benytte den samme kode til 4x4 sudokuer, hvilket både har forenklet noget af koden, men også kompliceret andre dele, hvis vi fx har været nødt

til at ændre noget, for at det virkede i Java eller for sudokuer i forskellige størrelser. Desuden er det er stadig ikke muligt at skabe større sudokuer end 9x9, bl.a. fordi nogle værdier bliver repræsenteret ved strenge, hvor der bliver trukket tal udfra, og dette kan ikke gøres hvis der kan forekomme tocifrede tal.

Konklusion

Ovenstående design og beskrivelse af implementationen er efter vores mening en godt sammenhængende og brugbar metode til at konstruere et sudokuspil for børn i 0.-3. klasse. Der er klare kvaliteter som opdelingen i MVC, den lave coupling og høje cohesion samt den mestendels velmodulerede kode. Dog kan denne forbedres ved at dele op i endnu flere mindre klasser for overskuelighedens og modulerings skyld.

Herudover har dette design opfyldt vores kravspecifikation på nær det ene krav med symbol-sudokuer, vi har valgt at undlade på baggrund af udtalelser fra lærere. Så alt i alt et tilfredsstillende design.

A.4. Testspecifikation

Testspecifikation

Formålet med dette dokument er at teste vores implementation af et Sudoku-spil til børn i 1.-3. klasse. Dette skulle gerne munde ud i at finde eventuelle fejl, samt at få målgruppens vurdering af kvaliteten af produktet; både ud fra brugervenlighed, men også funktioner, udseende og spilbarhed. Testen er delt op i 3 dele: modultest, systemtest og brugertest. De tre dele forklares uddybende herunder.

Modultest

Vores modultest er begrænset til at teste vores Game-model. Controller og View bliver testet under systemtesten i stedet, da disse ikke umiddelbart er indlysende at teste separat.

Game-modulet består af én constructor og 34 metoder. Da det ville tage det meste af en måneds arbejde at teste alle metoder i alle detaljer, har vi valgt at fokusere på den overordnede funktionalitet af Game-modulet i henhold til kravspecifikationen. I denne er det krævet, at følgende er opfyldt:

- Krav 1: Generering af valide 4x4- og 9x9-sudokuplader.
- Krav 3: Lav responstid.
- Krav 5: Hjælpesystem.
- Krav 7: Mulighed for flere sværhedsgrader.

Funktionstesten er lavet med fokus på at efterprøve, om disse krav kan opfyldes med den endelige kode. Testen i sig selv er udført i en enkelt JUnit TestCase (GameTest.java) indeholdende 6 metoder/tests. Der er brugt assertTrue og assertFalse undervejs i testen for at monitorere korrekt opførsel. De beskrives herunder i detaljer:

1. test4x4Validity():

Formål: At teste, om modellen genererer valide 4x4 sudoku-plader, og om modellen er i stand til at afgøre, om en plade er løst (dvs. dermed også, om den er valid).

Udførsel: Først genereres en udfyldt 4x4-plade (med felter fjernet), og det checkes, at pladen er af den rigtige størrelse og ikke er løst (eftersom der er udfyldte felter). Herefter genereres en tom plade, hvor vi manuelt indsætter en gyldig 4x4 løst sudoku. Vi beder så igen modellen om at afgøre om pladen er løst (hvilken den er!). Til sidst erstatter vi ét af tallene på den løste plade med et forkert og beder

igen modellen afgøre, om pladen er løst (hvilket den nu ikke er!).

Resultat: Testen er udført med succes.

2. **test9x9Validity():**

Denne test gør præcis det samme som ovenstående, blot for en 9x9-plade i stedet.

Resultat: Testen er udført med succes.

3. **test4x4GenerationTime():**

Formål: At måle, om modellen kan generere 4x4-plader inden for den i kravspecifikationen fastsatte tidsgrænse.

Udførsel: Her registreres en starttid, hvorefter der genereres 25 4x4-plader. Straks derefter registreres sluttiden, og en gennemsnitstid for generering af én plade sammenholdes med kravet om en responstid på under 2 sekunder.

Resultat: Testen er udført med succes.

4. **test9x9GenerationTime():**

Denne test gør præcis det samme som ovenstående, blot for en 9x9-plade i stedet.

Resultat: Testen er udført med succes.

5. **test4x4Help():**

Formål: At teste korrekt opførsel af de metoder, der er essentielle for, at hjælpefunktionen i spillet fungerer.

Udførsel: I testen udfyldes først en hel 4x4-plade og derefter sættes 3 forkerte tal ind på pladser, som modellen anviser os med `getUnsolvedField()`. Vi beder nu modellen finde alle forkerte tal på pladen (`getInvalidFields()`) og tjekker, om det er netop de 3, vi før just har indsat.

Resultat: Testen er udført med succes.

6. **test9x9Help():**

Denne test gør præcis det samme som ovenstående, blot for en 9x9-plade i stedet.

Resultat: Testen er udført med succes.

Alle tests er afprøvet på MacOS 10.5 og Windows XP. Samtlige tests er lykkedes alle gange. Man kan dog i teorien komme ud for, at `test9x9GenerationTime()` fejler på langsommere systemer, men dette har vi endnu ikke oplevet.

Det sidste krav om mulighed for flere sværhedsgrader er implementeret i modellen i form af mulighed for at fjerne flere felter på en plade, inden den vises for brugeren. GUT'et

er dog ikke lavet, så det understøtter flere sværhedsgrader på samme pladestørrelse, men man kan vælge mellem 4x4- (nemme) og 9x9- (svære) plader.

Alt i alt kan vi ud fra disse 6 tests konstatere, at de for modellen relevante krav fra kravspecifikationen er opfyldt.

Systemtest

I det følgende testes det på flere forskellige systemer, at programmet fungerer efter hensigten, dvs. at de handlinger, der kan udføres rent faktisk giver et korrekt resultat. Testen er altså både en systemtest, men til dels også en test af View-modulet og dets samspil med Control-modulet. Vi har valgt at sammenkøre disse tests, da en separat test af de to moduler ikke umiddelbart ville give mening.

Alle tests er udført på både MacOS 10.5, Windows XP, Windows Vista og Linux KDE/Gnome (OpenSuse 10.3). Eventuelle uoverensstemmelser mellem de forskellige styresystemer er noteret i konklusionen til den pågældende test. Herunder følger en nummereret skematisk gennemgang af tests af mulige udførbare handlinger i programmet. Det antages, at spillet i forvejen er installeret efter anvisningerne i installationsvejledningen. Ligeledes er de mange test så vidt muligt grupperet, så de er ens for hhv. 4x4- og 9x9-udgaverne. Eventuelle afvigelser her fra er bemærket i konklusionen.

Opstart:

#	Handling:	Forventet resultat:	Faktisk resultat:	Konklusion:
1	Opstart af spil med dobbeltklik.	Spillet starter op, og InitialDialog vises.	-II-	OK.
2	Vælg 4x4 sudokuplade på opstartsdialogen vha. billedet af en sådan.	Der genereres en 4x4 sudokuplade, og skærbilledet skifter til selve spillepladen.	-II-	OK.
3	Vælg 4x4 sudokuplade på opstartsdialogen vha. knappen under billedet.	Der genereres en 4x4 sudokuplade, og skærbilledet skifter til selve spillepladen. Timeren starter.	-II-	OK.
4	Vælg 9x9 sudokuplade på opstartsdialogen vha. billedet af en sådan.	Der genereres en 9x9 sudokuplade, og skærbilledet skifter til selve spillepladen. Timeren starter.	Linux: på visse distributioner vises i nogle tilfælde ikke en spilleplade. Alle andre systemer: samme som forventet resultat.	OK for alle andre systemer end visse distributioner af Linux, hvor problemet forekommer sporadisk.

Generering af nyt spil undervejs:

#	Handling:	Forventet resultat:	Faktisk resultat:	Konklusion:
5	Vis opstartsdialogen ved at klikke på "Nyt spil-knappen i 4x4- og 9x9-udgaven.	Opstartsdialogen vises i stedet for spillepladen.	-II-	OK.

Indsættelse og sletning af tal:

#	Handling:	Forventet resultat:	Faktisk resultat:	Konklusion:
6	Marker et tal på tasturet til højre (testet for samtlige tal + sletknappen), 4x4- og 9x9-udgaven.	Tallet markeres med en rød kant.	-II-	OK.
7	Indsæt det markerede tal på et udfyldt felt på pladen, uden at der løses en række/søjle/minigridd (testet for samtlige tal), 4x4- og 9x9-udgaven.	Tallet indsættes på pladen.	-II-	OK.
8	Forsøg at indsætte et tal på et gult (foruddefineret) felt.	Der sker intet.	-II-	OK.
9	Forsøg at slette et tomt felt.	Der sker intet.	-II-	OK.

Feedback:

#	Handling:	Forventet resultat:	Faktisk resultat:	Konklusion:
10	Indsæt et korrekt tal på et udfyldt felt på pladen, så der løses en række/søjle /minigridd (testet for samtlige tal), 4x4- og 9x9-udgaven.	Tallet indsættes på pladen, den/de løste række/ søjle /minigridd blinker grønt og farves derefter gule.	-II-	OK.
11	Indsæt et korrekt tal på det sidste udfyldte felt på pladen, så pladen løses.	Hele pladen blinker grønt, tilbage til gult og derefter blinker minigridds kontinuertligt i forskellige farver. Desuden gøres talknapperne og vink-knappen grå og kan ikke klikkes på. Timeren stoppes.	-II-	OK.

Hjælpesystem:

#	Handling:	Forventet resultat:	Faktisk resultat:	Konklusion:
12	Klik på Vinkknappen, når der er forkert indsatte tal på pladen, 4x4- og 9x9-udgaven.	De forkerte tal blinker rødt og returnerer derefter til hvid. Desuden inkrementeres timeren med 20 sekunder, og antal vink sættes op med 1.	-II-	OK.
13	Klik på Vinkknappen, når der ingen forkert indsatte tal er på pladen, 4x4- og 9x9-udgaven.	Et endnu ikke udfyldt felt får indsat dets korrekte værdi, blinker grønt og returnerer derefter til hvid og får værdien fjernet igen. Desuden inkrementeres timeren med 20 sekunder, og antal vink sættes op med 1.	-II-	OK.
14	Klik på Vinkknappen, når pladen er løst.	Der sker intet.	-II-	OK.

Afslutning af programmet:

#	Handling:	Forventet resultat:	Faktisk resultat:	Konklusion:
15	Afslut spillet fra opstartsdialogen ved at klikke på døren i nederste venstre hjørne.	Programmet afsluttes.	-II-	OK.
16	Afslut spillet fra spillepladen ved at klikke på døren i nederste venstre hjørne, 4x4- og 9x9-udgaven.	Programmet afsluttes.	-II-	OK.

Konklusion på systemtest

På nær et sporadisk problem med test nummer 4 er systemtests gennemført med succes. Vi kan herud fra konkludere, at programmet svarer korrekt på brugernes handlinger.

Brugertest

Indledende forberedelse af brugertest

Sted: Mariendal Friskoles SFO

Tid afsat: Ca. 1 time.

Indledende forklaring:

For at lave et rigtig godt computerspil vil vi gerne vide, hvad I synes om det, så vi kan få det til at blive bedre og nemmere at spille for alle børn i fritidsordninger.

Testforløb:

- Pædagog installerer programmet på en computer og giver feedback på eventuelle problemer.
- Pædagog forklarer børnene, hvordan de skal gøre (opstart og regler) (5-10 min.).
- Børnene spiller spillet (20-30 min.).
- Vi observerer og skriver ned.
- Undervejs stiller vi børnene spørgsmål, se nedenstående.
- Bagefter sætter vi os sammen med børnene og snakker om spillet (10-15 min.).

Ting vi vil være opmærksomme på:

- Hvor hurtigt kommer de i gang? (Har de problemer med InitialDialog?)
- Hvor stort er behovet for hjælp til at komme i gang? (Behovet for hjælp skulle jo gerne minimeres).
- Hvad klikker de på?
 - Forsøger de at ændre noget, der ikke kan ændres?
 - Kan de finde ud af at sætte tal ind?
- Holde øje med deres reaktioner: Synes de det er spændende eller kedeligt.
- Forsøger de drag'n'drop?
- Hvordan er reaktionen på feedback? Forstås det, hvad der sker?

Spørgsmål til børnene under spillet:

- Hvordan skal man indsætte et tal?
- Hvordan ville du slette et tal?
- Hvis du nu ikke kunne finde flere tal, hvad ville du så gøre?
- Hvis du ville afslutte spillet, hvad ville du så klikke på nu? (For både BoardViewer og InitialDialog)
- Hvis du gerne ville have en anden sudokuplade, hvad ville du så gøre?
- For mange/få ord? Forstår du, hvad der står her?

Spørgsmål til børnene efter spillet:

- Var det let eller svært at starte spillet?
- Hvad var sværest ved spillet?
- Hvad var nemmest ved spillet?
- Hvad var bedst? Hvorfor?
- Hvad var mest irriterende? Hvorfor?
- Var det sjovere end at løse dem på papir? Hvorfor?
- Hvor mange synes, det gik for langsomt/hurtigt?
- Hvor mange synes, det var for let/svært?
- Hvor mange så på tiden, når de løste opgaverne? Og synes, det var godt/skidt med tid?
- Hvor mange fik et vink?
- Hvis de synes, det er kedeligt, synes de så også, at det er kedeligt i forhold til matematikundervisning/løsning på papir?

Spørgsmål til lærerne/pædagogerne:

- Hvad var godt/skidt ved installations- og brugsvejledningen? Var den let at læse og forstå?
- Hans/hendes indtryk af, om børnene kunne finde ud af det. Herunder
 - Var det for let/svært? At spille? At starte op?
 - Hvor meget bedre er det at løse dem på computer i forhold til papir. Herunder om hjælpefunktionen aflastede.

Spørgsmål: Må ikke være ja/nej spørgsmål.

Forløb af brugertest

Brugertesten blev gennemført mandag den 2. juni med 8 børn fra Mariendal Friskole i København. Børnene fordelte sig med 1 barn fra 1. klasse og 7 fra 2. klasse. Vi havde egentlig fået 2 børn fra hvert klassetrin (0.-3.), men en del var allerede taget hjem.

Før selve testen

- Pædagogerne havde desværre ikke tid til at overvære testen. Derfor måtte vi selv installere programmet på 4 computere (kopiere filen over fra CD-ROM), der alle kørte Windows XP og havde Java Runtime Environment (JRE) installeret. Derefter startede spillet op uden besværligheder med et dobbeltklik.
- Vi stod derfor også selv for instruktionen af eleverne, der alle før havde prøvet at løse sudoku-plader. Børnene fik derfor kun at vide, at spillet lå midt på computerens skrivebord, og at de skulle begynde med at løse den lille sudoku.
- De 8 børn blev bedt om at sætte sig i par, da vi desværre ikke havde nok computere til alle. Vi havde forinden overvejet dette i forhold til at sætte dem selv ved hver deres computer. Vi mente, at hvis børnene sad to og to, ville de snakke mere sammen, og vi ville generelt have lettere ved at overhøre deres konversation og tanker. Vi mener ligeledes, at dette (i hvert fald i en undervisningssituation) vil være det mest sandsynlige scenarie pga. begrænsede edb-faciliteter.

Under selve testen

- Samtlige børn startede af sig selv og uden instruktion meget hurtigt spillet med et dobbeltklik.
- Børnene indså hurtigt, hvilken af pladerne på opstartsdialogen, der var den svære, og hvilken der var den lette. Nogle havde dog problemer med at klikke sig ind på pladerne, men dette virker som et sporadisk problem.
- Nogle af børnene forsøgte sig med det samme med drag'n'drop for at indsætte tal. Da dette ikke virkede, forsøgte nogle børn andre metoder (som så virkede), mens en enkelt gruppe til sidst måtte have lidt hjælp til at få sat tal ind.
- Børnene fandt selv ud af, hvordan spillets andre funktioner virkede. F.eks. kunne de selv regne ud, hvordan man fik en ny plade og afsluttede spillet. Da vi sagde til dem, at man kunne få hjælp (men ikke hvordan!), vidste de med det samme, hvordan man fik det.
- Nogle grupper misforstod, at selv om man sætter et rigtigt tal ind, får man ikke nødvendigvis feedback (kun hvis der er løst en korrekt række/søjle/minigrid). Til gengæld begyndte flere af grupperne selv at diskutere, hvordan spillet skulle spilles og lærte dermed af hinanden. De fandt derfor allesammen ud af, hvordan feedbacksystemet fungerede.

- Børnene var ret begejstrede for den grønne feedback, og det var specielt ”fedt med sådan noget disco-noget til sidst” (når en plade var løst).
- Flere grupper valgte at slutte spillet og starte det igen for at generere en ny plade. Det virkede dog ikke umiddelbart som om, at det var uintuitivt for dem at gøre det på denne måde.
- Flere børn tog forholdsvist lang tid om at gennemskue Vink-knappens funktion i de to tilfælde. Men da funktionen var kendt, blev den brugt efter hensigten.
- Få af børnene opdagede timeren under brættet. Men da den første gruppe havde opdaget den, gik der pludselig en lille smule sport i at have den bedste tid.
- Det virkede ikke som et problem, at programmet skifter frames mellem opstartsdialogen og selve pladen (i et kort øjeblik vises skrivebordet mellem skiftene). Computerne var hurtige nok til, at generering af selv de store plader tog under 1 sekund.
- Nogle af børnene blev forvirrede over, at de kunne indsætte et tal, selvom der ikke var valgt et tal på ”tastaturet”. Dette må tilskrives en bug i koden (se nedenfor).
- Ingen af børnene havde problemer med at spille med tal (de kunne alle genkende og anvende både tallene fra 1-4 og 1-9).

Efter testen

- Børnene syntes, at 4x4-pladerne var for nemme. 9x9-pladerne var svære, men ikke uoverkommelige. Flere af grupperne havde nået at løse en hel 9x9-plade uden at have klikket mange gange på Vink-knappen inden for 20-25 minutter.
- Børnene foreslog, at man kunne lave en mellemting mellem 4x4 og 9x9, eller at man kunne få mulighed for at vælge sværhedsgrader på de enkelte plader.
- To børn kommenterede, at spillet blev for nemt med feedback-systemet, mens andre synes, at ”det var rigtig fedt”. De, som syntes, det blev for nemt, skal for en god ordens skyld lige placeres i kategorien af mere rutinerede sudoku-spillere.
- Børnenes kommentar til, hvad der var det bedste ved spillet, var, at det var spændende at løse opgaver. Flere mente, at ”det hele” var det bedste ved spillet.
- Nogle børn havde brugt Vink-knappen til at klikke sig igennem hele spillet, fordi det var sjovt. Andre nægtede at nedværdige sig til at bruge Vink-knappen. Børnene foreslog selv flere alternative måder at bruge Vink-knappen på:
 - Man kunne gøre den inaktiv, når der var mindre end 5-10 uløste felter tilbage på pladen.
 - Man kunne gøre sådan, at knappen kun kunne bruges et vist antal gange i løbet af et spil eller en bestemt tidsperiode.
 - Man kunne ”optjene” Vink ved f.eks. at løse en række/søjle/minigrid korrekt.

- De syntes generelt, at det var en god idé med timeren, da man så kunne udfordre sig selv og forbedre sig. Det var liegeledes en fordel, at tiden ikke talte ned i stedet for op, samt at det var retfærdigt, at der blev lagt tid til, når man fik et vink.
- Børnene syntes helt bestemt, at det var bedre og sjovere end at løse sudokuer på papir. Blandt andet fordi man kan få vink og slette tal nemt.
- Børnene var enige om, at ”hvis I laver flere computerspil, skal I komme herud og prøve dem også!”.

Problemer konstateret ud fra testen

1. Vi har konstateret et generelt problem, når brugeren forsøger at klikke et tal ind på pladen eller vælge pladestørrelse i opstartsdialogen. Klikket registreres ikke altid første gang.

Løsningsforslag: Dette har noget at gøre med vores MouseListeners’ metoder i de to klasser InitialDialog og BoardViewer. Et klik, hvor markøren flyttes under klikket, bliver ikke altid registreret. Dette er nu løst ved at implementere en `mousePressed()`-metode som supplement til `mouseClicked()`-metoden.

2. Børnene forsøgte sig i starten uden held med drag’n’drop for at indsætte tal.

Løsningsforslag: Dette har vi ikke umiddelbart planer om at implementere, men det ville være oplagt til en eventuelt senere udgave af programmet.

3. 4x4-pladerne var for nemme, og begge størrelser plader måtte gerne kunne varieres i sværhedsgrad.

Løsningsforslag: Det første er løst ved at skrue op for antallet af felter, der bliver fjernet i 4x4-pladerne (fra før mellem 6 og 10). Det andet kan først løses i en senere udgave af spillet, hvor en ekstra dialog kan tilføjes, hvor man kan vælge sværhedsgrad for den enkelte pladestørrelse.

4. Når der indsættes et tal, bliver det ikke nødvendigvis gult.

Løsningsforslag: Misforståelsen opstod efter vores mening, fordi børnene startede med de forholdsvis lette 4x4-plader, hvor man nærmest ikke kan undgå at få grøn feedback, hver gang man indsætter et tal. Når sværhedsgraden af 4x4-pladerne sættes op, vil der komme mindre grøn feedback.

5. Flere børn genererede en ny plade ved at slutte spillet og starte det igen.

Løsningsforslag: Ny-knappen i BoardViewer-klassen skal gøres mere intuitiv, så børnene nemmere kan gennemskue dens funktion. Der er blevet tilføje et ikon, der er mere sigende (en uløst 4x4-plade) teksten er ændret til ”Nyt spil” i stedet for bare ”NY”. Desuden har Vink-knappen fået en lysende pære tilføjet i stedet for det mere kedelige spørgsmålstegn.

6. Der blev fundet en bug, hvor man kunne sætte tal ind, selvom der ikke var trykket på nogle af spillets toggle buttons.

Løsningsforslag: Når man undertogler en talknap, skal `insertValue` i BoardViewer sættes til -1. Controlleren skal så tjekke værdien af `insertValue`, inden den indsætter tallet i Gamet. Dette er nu rettet.

7. Timeren blev først opdaget meget sent i forløbet.

Løsningsforslag: Timeren er gjort større og er rykket længere ud til venstre. Desuden er der tilføjet en tæller for det antal vink, man har brugt i løbet af spillet til højre for timeren.

8. Vink-knappens funktion tog tid for børnene at forstå helt.

Løsningsforslag: Vink-funktionaliteten er ændret, så de eventuelt forkerte tal blot markeres med rødt (ikke fjernes) i lidt længere tid. På samme måde markeres et nyt tal grønt i lidt længere tid og forsvinder så igen. Det bliver altså ikke stående. På denne måde hjælper den hverken for meget eller gør det umuligt for børnene at indse, hvad der var galt.

Udvalgte citater fra brugertest

- "Meget sjovt spil!"
- "YES!" (udbrud fra spiller, første gang han fik feedback)
- "God ide at man kan få hjælp."
- "Øv, den gi'r os bare et nyt tal..." (kommentar til Vink-knappen)
- "Nu skal vi jo ikke gætte!" (et par begynder at løbe tør for muligheder for at indsætte tal)
- "Det er jo en genialitet!" (en spiller kommenterer på Vink-knappens funktionalitet)
- "Det hele var godt!" (til spørgsmålet om, hvad der var bedst ved spillet)
- "Fedt med sådan noget disco-noget til sidst." (kommentar til pladens udseende, efter man har løst den)

Konklusion på brugertest

Brugertesten afslørede flere forbedringsforslag, hvor de, der kunne nås, er blevet implementeret efterfølgende. Testen afslørede også en enkelt bug i programmet, som er blevet rettet.

Test af kravspecifikation

1. *Algoritme til generering af valide sudokuer*

Dette krav er testet i funktionstestens test 1 (for 4x4) og test 2 (for 9x9). Testen er kørt med succes, og kravet er dermed opfyldt.

2. *Simpel grafisk brugerflade*

Dette krav er indirekte testet i brugertesten. Udover de ovenfor nævnte problemer, kunne målgruppen nemt finde ud af bruge spillet, og derfor kan vi tillade os at konkludere, at kravet er opfyldt.

3. *Lav responstid*

På samtlige testede computere er responstiden under 2 sekunder på alle handlinger på nær opstart. Ligeledes er generering af sudokuplader testet i funktionstestens test 3 og 4, der begge er kørt et stort antal gange uden at fejle. Derfor er kravet opfyldt.

4. *Pointsystem*

Dette krav er brugertestet i form af den på pladen viste timer. Efter målgruppens respons at dømme, virker timeren efter hensigten. En high score-liste er ikke blevet implementeret, men en sådan var heller umiddelbart efterspurgt af målgruppen. Timeren er integreret med hjælpesystemet, men mangelen på en high score-liste gør, at kravet kun delvist er opfyldt.

5. *Hjælpesystem*

Dette er testet både ved brugertests og funktionstestens test 5 og 6. Målgruppen kunne efter kort tid anvende hjælpesystemet korrekt, ligesom funktionstesten viste, at funktionerne virker korrekt. Kravet anses derfor som værende opfyldt.

6. *Nem installation og afvikling*

Dette er endnu ikke testet af målgruppen (lærere eller pædagoger), men vi installerede selv programmet for målgruppen ved at kopiere filen over fra cd'en og dobbeltklikke for at starte. Desuden er der lavet en kort et-sides installationsvejledning. Programmet er testet på Mac, Windows (XP og Vista) og Linux (OpenSuse 10.3). Kravet anses derfor som værende opfyldt.

7. *Mulighed for flere sværhedsgrader*

Dette krav er delvist implementeret, da brugeren har mulighed for at vælge mellem 4x4- eller 9x9-plader. Modellen i dens nuværende udseende understøtter desuden forskellige sværhedsgrader inden for disse to kategorier, men af tidsmæssige og pædagogiske årsager har vi valgt ikke at implementere det i GUI'et. Vi henviser til rapporten for nærmere begrundelse.

8. *Mulighed for at spille med tal eller symboler*

Dette krav er blevet fraveget efter samtale med forskellige skolelærere og er blevet bekræftet efter brugertesten, idet børnene kunne tallene og ingen problemer havde med at anvende dem, selv i første klasse. Vi henviser igen til rapporten for en nærmere begrundelse.

9. *Feedback*

Et feedback-system er blevet implementeret og afprøvet på målgruppen. De problemer, der dukkede op under brugertesten, er blevet rettet, og kravet kan derfor anses som værende opfyldt.

Samlet konklusion på tests

Alle tests er blevet gennemført og evalueret. De fejl og mangler, der er blevet opdaget under testen, er enten blevet rettet eller dømt for tidskrævende til at kunne implementere. Alt i alt fungerer programmet efter hensigten, både med hensyn til den underliggende "motor" og med hensyn til at reagere på handlinger fra brugeren.

A.5. Installations- og brugervejledning

Installationsvejledning

Systemkrav

Programmet er garanteret kompatibelt med Windows XP og Vista, Mac OS 10.5 og Linux openSUSE 10.3 KDE, og burde generelt køre på Mac, Windows eller Linux-versioner, der er Java Runtime Environment (JRE) 5.0 kompatible.

Java Runtime Environment 5.0 eller senere versioner skal være installeret, før programmet kan køre, se nedenfor.

Windows og Linux

Hvis der er installeret Java Runtime på din computer skal du:

1. Kopiere filen fra CD-rommen.
2. Dobbeltklikke på filen for at køre spillet.

Ellers skal Java Runtime downloades fra:

<http://www.java.com/en/download/index.jsp>.

Vælg den version, der hører til dit styresystem, dvs. vælg om det er en Windows eller Linux computer. Det er gratis at downloade Java.

Hvis du er i tvivl om hvorvidt JRE er installeret på din computer prøv da først at dobbeltklikke på spillet, og se så nedenstående problemløsning.

Mac

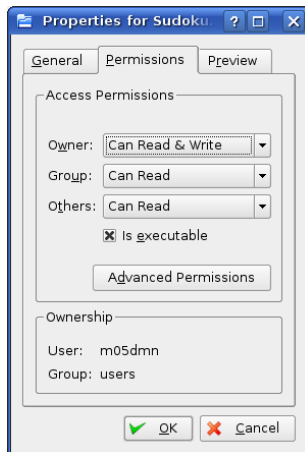
Der er installeret Java Runtime på din computer, så du skal:

1. Kopiere filen fra CD-rommen.
2. Dobbeltklikke på filen for at køre spillet.

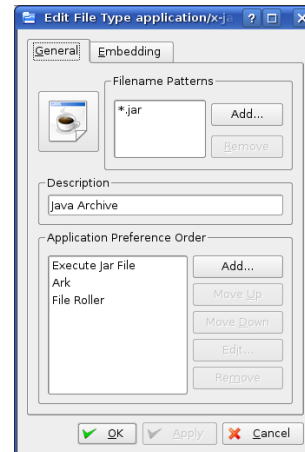
Problemløsning

- **Fejl:** Der åbner en dialogboks om hvorvidt man vil installere nogle filer.

Løsning: I stedet installeres Java Runtime Environment fra ovenstående link.



Figur 14: Under fanebladet Permissions sættes kryds ved "Is executable".



Figur 15: I menuen Type sættes "Execute jar file" øverst.

- **Fejl:** Meddelelsen "Sudoku.jar does not have execute permission" kommer frem (Linux KDE).

Løsning:

- Højreklik på Sudoku.jar og vælg Egenskaber/Properties.
- Under fanebladet "Permissions" skal der gives tilladelse til at være executable, dette gøres for det meste ved at sætte hak/kryds i feltet ved "Is executable". Se Figur 14.

- **Fejl:** Når der dobbeltklikkes åbnes en mappe i stedet for spillet (Linux KDE).

Løsning:

- Højreklik på Sudoku.jar og vælg Egenskaber/Properties.
- Under fanebladet "General" vælges "Type". Her sættes "Execute jar file" øverst af de mulige. Se Figur 15.

Generelt kan problemer ved dobbeltklik i Linux afhjælpes ved at køre spillet fra en terminal i mappen med Sudoku.jar ved hjælp af følgende kommando:

```
java -jar Sudoku.jar
```

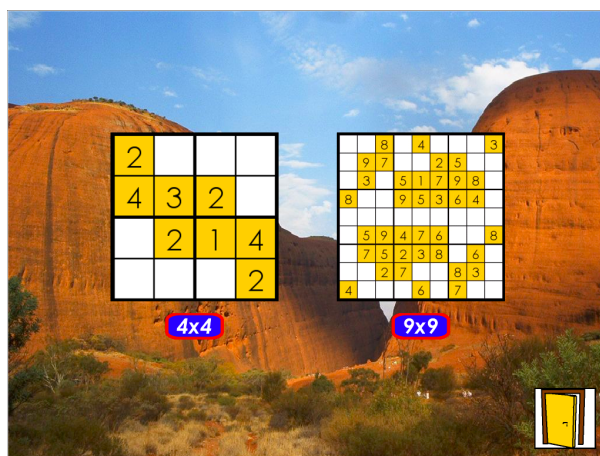
Brugsvejledning

Spilleregler for Sudoku

Spillet går ud på at sætte tal ind på pladen, så der i hver række, søjle og lille firkant præcis én gang står hvert tal fra 1-4 eller 1-9 (hvis man spiller en 4x4 hhv. 9x9 sudoku). Pladen er løst, når alle tal er sat ind korrekt.

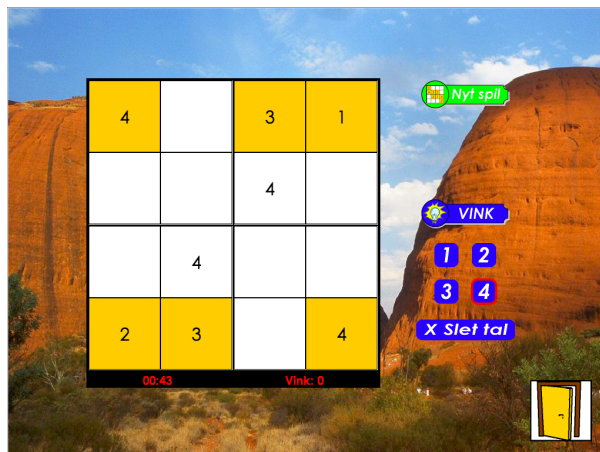
Hvordan spilles computerspillet?

1. Dobbeltklik på sudoku-ikonet.
2. Når startmenuen vises vælges den ønskede sværhedsgrad, 4x4 sudoku eller 9x9 sudoku.



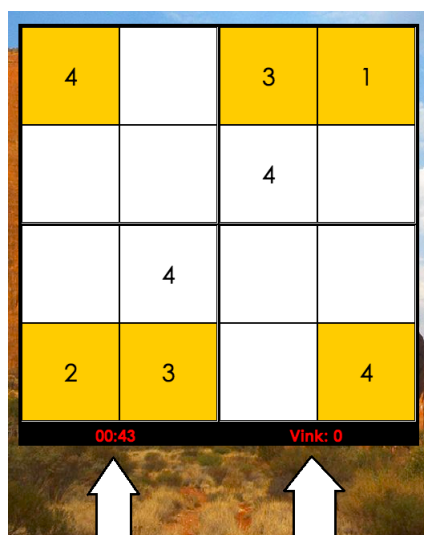
Figur 16: Startmenuen

3. Spillet kommer frem (se Figur 17) og du kan nu
 - *sætte et tal ind* på et hvidt felt på pladen ved først at klikke på det ønskede tal og så på det felt, hvor det skal indsættes. Hvis en række/søjle/lille firkant bliver løst korrekt ved et træk, blinker den grønt og farves derefter gul. Så kan man ikke rette mere i den.
 - *fjerne et tal* på et hvidt felt ved først at klikke på "Slet tal" og så på det tal, der skal fjernes. Det er kun tallene på hvide felter, der kan slettes.
 - *få et vink* ved at klikke på "VINK". Hvis der findes forkert indsatte tal på pladen, markeres disse med rød i et kort øjeblik. Ellers blinker et uløst felt grønt, mens det viser, hvilket tal der skal stå på dette felt. Samtidig med, at vinket ses, lægges ekstra tid til timeren, og antallet af vink øges.
 - *starte et nyt spil* ved at klikke på "Nyt spil".



Figur 17: Spillepladen

4. Timeren under spillet viser hvor lang tid, der er gået, siden dette spil begyndte (se Figur 18). Hvis man får et hint, lægges der ekstra tid til timeren og antallet af brugte vink øges.



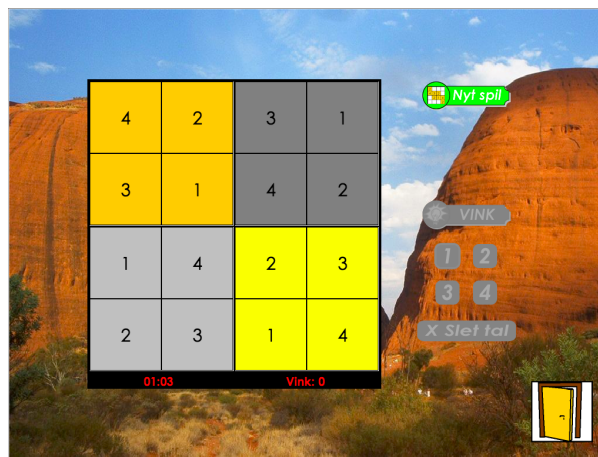
Figur 18: Timeren

5. På ethvert tidspunkt af spillet kan det afsluttes ved at klikke på døren (Figur 19).



Figur 19: Afslut spillet ved at klikke på døren.

6. Når et spil er løst, blinker de små firkanter i forskellige farver, indtil der klikkes "Nyt spil", eller spillet afsluttes (se Figur 20).



Figur 20: Sudokuen er løst og blinker i forskellige farver.

A.6. Mødereferater

Tirsdag, 22. april

Referatdato: 22. april

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 13.15-14.30
- Mødeleder: Ditte
- Referent: Anders

Vores forskellige personlighedstests blev gennemgået, og vi fandt ud af Ditte var mest forskellig fra Hjalte og Anders og derfor rent teoretisk ville være bedst som leder. Dette blev dog ikke besluttet.

Derefter gennemgik vi gruppemedlemmernes forskellige kompetencer inden for både programmeringssprog og andre relevante erfaringer. Vi besluttede på dette grundlag, at et projekt skrevet i Java erfaringsmæssigt ville være det mest farbare.

Herefter diskuterede gruppen de forskellige projektemner. Visse projekter blev udelukket pga. manglende kompetencer (projektdatabasen og DIKUTAL-projektet). Gruppens rent matematiske baggrund gjorde, at sudoku-projektet virkede mest tiltrækkende, hvorfor dette fik første prioritet.

Tirsdag, 29. april

Referatdato: 29. april

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 11.00-12.00 og 13.00-16.00
- Mødeleder: Anders
- Referent: Hjalte

Vi havde alle gjort os flere tanker omkring vores spil, så for at få det hele ned på papir gennemførte vi på forslag af Anders en brainstorm med vores sudokuspil som emne.

Vi kom op med:

- Algoritme til at generere sudokuspil
- Størrelser på spilleplader

- Udseende - simpel brugerflade, mulighed for at spille med symboler eller farver i stedet for tal
- Mulighed for forskellige niveauer
- Hjælp til brugerne - hhv. indbygget i spillet og hvilke krav vi kan sætte til lærerne
- Formelle krav - Responstid, portabilitet osv.
- Pointsystem - Highscoreliste evt. afhængig af brug af hjælpefunktion

Herefter diskuterede vi vores ideer for at afgøre hvad der er nødvendigt, muligt, fornuftigt og/eller belejligt at have med i spillet, og hermed dannet os et overblik over hvad vores kravspecifikationer skal indeholde.

Efter at have diskuteret vores ideer med instruktør og andre grupper til øvelsestimen, har vi præcist formuleret vores kravspecifikationer. Vi mangler dog stadig at:

- beskrive baggrund og formål med projektet (laver Ditte til fredag)
- lave et diagram over kravspecifikationerne (laver Anders til fredag)

Onsdag, 7. maj

Referatdato: 7. maj.

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 14.30-17.00
- Referent: Ditte

Timer brugt siden sidst:

Anders: 1 time på kravspecifikation, 1/2 time på skema, 2 timer på at læse i bogen.

Ditte: 1/2 time på kravspecifikation, 1 time på informationssøgning på nettet.

Hjalte: 1/2 time på referat, 1 time på kravspecifikation.

Dagsorden:

1. Rette i kravspecifikation i forhold til kommentarer fra instruktøren.
2. Begynde på baseline design.
3. Synkronisere kalendre.

Vi rettede i kravspecifikation ud fra Zahra's kommentarer, så det blev mere overskueligt og vores krav ikke blev for brede, men derimod "målelige".

Vi overvejede den bedste datastruktur til vores algoritme og blev foreløbigt enige om todimensionelt array.

Vi aftalte overordnet hvilke dage vi kan mødes indtil projektaflevering.

Anders tegnede og fortalte mens vi diskuterede hvordan moduldesignet skal se ud.

Til næste gang:

Ditte gør kravspecifikationen helt færdig.

Vi mødes næste gang i morgen, torsdag d. 8. maj.

Torsdag, 8. maj

Referatdato: 9. maj

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 18.30-22.40
- Referent: Ditte

Timer brugt siden sidst:

Anders: 2 timers læsning i bogen.

Ditte: Ingen.

Hjalte: Ingen.

Dagsorden:

1. Fortsætte beskrivelsen af baseline design.
2. Finde ud af hvordan vi vil lave vores algoritme.

Vi fortsatte diskussionen om opbygningen af moduldesignet og lagde os fast på nogle forskellige ting.

Vi søgte på nettet for at hente inspiration til vores algoritme til at generere sudoku-pladerne. Vi fandt forskellige informationer og kode til sudokuløsere. Desuden hentede vi "Programming Sudoku" som e-bog.

Vi blev enige om at der grundlæggende er to muligheder for generering af sudoku-plader, som vi vil se nærmere på:

- At udfylde en sudokuplade ved at begynde med et felt, vælge et tilfældigt tal mellem 1 og 9, indsætte det. I næste felt indsættes et tilfældigt af de mulige tal og således fortsættes indtil pladen er udfyldt eller et felt ikke kan udfyldes. Så går man et skridt tilbage og indsætter et andet tal og prøver om dette virker osv. Fordele: Hvert forsøg giver en udfyldt sudokuplade. Ulemper: Der kan forekomme megen backtracking.
- At udfylde en sudokuplade ved at angive de mulige tal, der kan stå i hvert felt, vælge et tilfældigt felt og indsætte et tilfældigt af de mulige tal. Derefter fjernes denne mulighed fra alle felter i samme række, søjle og kvadrat. Så vælges et nyt felt blandt dem, der har færrest muligheder tilbage og processen fortsættes. Fordele: Ingen backtracking, og stor sandsynlighed for succes. Ulemper: Det kan tage flere forsøg at skabe en helt udfyldt plade.

Herefter fjernes felter et par stykker af gangen indtil det ønskede antal tomme felter opnås. I hvert skridt tjekkes vha. en sudokuløser om sudokuen kan løses. Ellers sættes tallene ind igen og nogle andre fjernes i stedet.

Absalonlog

Vi har ikke haft problemer med at logge på Absalon i denne uge.

Mandag, 12. maj

Referatdato: 12. maj.

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 9.00-13.00
- Referent: Anders

Timer brugt siden sidst:

Anders: 1 times læsning i "Programming Sudoku" og en halv times klassediagram.

Ditte: Ingen.

Hjalte: 1,5 timers læsning.

Dagsorden:

1. Færdiggøre sidste to dele af baseline design.
2. Oversættelse af bogens metoder, så de er kompatible med vores design.

Vi brugte forholdsvis lang tid på at afgøre, hvilken implementation af sudoku-generatoren, vi ville bruge. Vi havde valget mellem to mulige, efter at have søgt nettet tyndt i en times tid:

- Bogen "Programming Sudoku", af Wei-Meng Lee (se nærmere på Absalons diskussionsforum), der indeholder en komplet gennemgang af teori og praksis for generering og løsning af sudokuer.
- "Java Sudoku": en open-source Java-implementation af et sudoku-spil (<http://playsudoku.sourceforge.net>).

Vi gennemså de to muligheder og besluttede os til sidst for at bruge den første, da der her var teori og kommentarer til koden, ligesom det ville være væsentlig mere kompliceret at skulle få de to løsninger til at hænge sammen. Desuden er open source implementationen ikke kommenteret og teorierne bag er heller ikke umiddelbart tilgængelige.

Udfordringen i den valgte løsning ("Programming Sudoku") bliver så at få oversat koden fra Visual Basic 2005 til Java. Dette har vi valgt at se som en overkommelig opgave, da de to sprog ligner hinanden forholdsvis meget (de er begge objektorienterede højniveausprog, og den rå databearbejdning er let at konvertere). Vi fik desuden afgrænset problemet, idet vi i vores baseline kun skal generere forholdsvis lette sudokuer, dvs. vi ikke skal lade programmet forsøge avancerede løsningsmetoder. Vi skal ej heller bruge muligheden for at gemme og åbne spil eller fortryde/genskabe træk.

Jævnfør diskussionen om generering af sudokuplader på sidste møde vil vi derfor nu anvende bogens metode, der bruger brute-force til at generere en udfyldt plade, hvorefter en symmetrisk spilleplade genereres ved at fjerne to felter ad gangen og forsøge at løse sudokuen med de simple metoder.

Til næste gang:

- Ditte sætter sig ind i bogens generator-metoder.
- Hjalte sætter sig ind i bogens løser-metoder.
- Anders laver oplæg til GUI-design, skriver mødereferat og læser overordnet på SudokuPuzzle-klassen fra bogen.

Næste møde: tirsdag d. 13. maj.

Tirsdag, 13. maj

Referatdato: 14. maj.

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 11.00-23.00
- Referent: Anders

Timer brugt siden sidst:

Anders: 3 timer design af GUI og klassestruktur (inkl. læsning i bogen)

Ditte: 2,5 timers gennemlæsning af bogens kode.

Hjalte: 2 timers gennemlæsning af bogens kode.

Dagsorden:

1. Færdiggøre baseline design.
2. Afholde kort møde med Zahra (vores instruktør)
3. Aftale nærmere forløb for baseline kodning.

Mødet begyndte med en kort gennemgang af, hvad der skulle nås i løbet af dagen. Vi fandt ud af, at vi havde forholdsvis travlt, da et færdigt baseline design skulle afleveres senest om aftenen. Arkitektur-delen af designet var færdiggjort på mødet inden, derfor skulle moduldesign og algoritmer og datastrukturer laves færdigt, ligesom der skulle argumenteres for designets fordele/ulemper, anvendte designmønstre, coupling/cohesion, abstraktion og lignende.

Derefter gennemgik Anders forslag til foreløbigt GUI-design og skitse af klassestruktur. Begge blev godkendt af gruppen.

Til øvelsestimerne fortsattes diskussionen om, hvorledes bogens implementation kunne konverteres til vores brug. Vi fik identificeret de metoder, der *ikke* skulle bruges, og efter et halv times møde med Zahra (vores instruktør), fik vi positiv respons på vores planer for baseline design, GUI-design og opbygningen af projektet som helhed. Zahra viste og ligeledes et godt baseline design fra et tidligere kursus, der gav os frisk mod på, at vi ikke behøvede at skrive 20-30 sider, men kunne nøjes med en kortere og mere kompakt beskrivelse af designet.

Zahra kommenterede ligeledes vores mødereferater, som hun overordnet syntes, var gode. Hun kunne dog godt bruge en lidt større udpensling af beslutninger og diskussioner, hvilket vi vil forsøge at efterleve fremover. Vi diskuterede også tidsforbruget på projektet generelt og udtrykte vores forundring over, at der var to ugers forberedelse til den mundtlige fremlæggelse i stedet for at udskyde afleveringsfristen for rapporten med 5 dage eller en uge. Zahra ville tage dette med videre.

Efter mødet gik vi for alvor i gang med baseline designet. Ditte og Hjalte gav sig til at nærlæse bogens implementation af sudoku-genereringsalgoritmen og skrive pseudokode for denne. Anders gav sig i kast med selve baseline-designets modulopbygning, overvejelser omkring GUI-design og generelle fordele/ulemper ved designet. Det blev besluttet så vidt muligt at lægge sig op ad bogens implementation og datastrukturer. Efter mange lange timer blev der fælles gennemlæst og skrevet indledning og konklusion, hvorefter designet blev afleveret.

I løbet af dagen fik vi ligeledes uploadet en revideret kravspecifikation, hvor diagrammet bedre afspejlede baseline krav kontra senere krav, kildeinformationen til hvert krav blev specificeret nærmere, og kravet om ”indbydende design” blev erstattet af mere målbare krav.

Til næste gang:

- Intet, da vi mødes igen om formiddagen dagen efter (vi skal jo også sove!).

Næste møde: onsdag d. 14. maj.

Onsdag, 14. maj

Referatdato: 15. maj.

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 11.00-15.00
- Referent: Anders

Timer brugt siden sidst:

Ingen.

Dagsorden:

1. Skrive mødereferat for den 13. maj
2. Begynde oversættelse af kode fra bogen.

Anders skrev referat fra dagen før, mens Ditte og Hjalte påbegyndte oversættelsen af koden fra bogens SudokuPuzzle-klasse. Til sidst hjalp Anders også med dette. Vi fik på dagens møde oversat ca. 75% af algoritmen fra bogen, resten blev opdelt blev det aftalt at gøre selvstændigt derhjemme. Oversættelsen gik særdeles smertefrit, men testningen

skal overvejes grundigt, ligesom 1-2 af os skal i gang med GUI'en snarest.

Til næste gang:

- Færdigoversættelse af koden fra bogen.

Næste møde: fredag d. 15. maj.

Fredag, 16. maj

Referatdato: 16. maj.

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 9.00-12.00
- Referent: Anders

Timer brugt siden sidst:

Ingen.

Dagsorden:

1. Aflevere mødereferater for uge 20.
2. Færdiggøre oversættelse af kode.
3. Igangsættelse af GUI-design og kodning heraf.

Dagen startede med at få et overblik over manglende metoder fra bogens kode. Derefter færdiggjordes kodningen, og Hjalte begyndte på brugerinterfacedesign i NetBeans ud fra skitsen i baseline designet. Anders skrev referat og afleverede disse for uge 20.

Til næste gang:

- Anders kontakter skoleleder med spørgsmål ang. afprøvning på målgruppe og indledende spørgsmål omkring børnenes perceptionsevner og kompetencer mhp. effektiv og målrettet strukturering af GUI'et.
- Ditte samler gruppens kode til én klasse og leder efter manglende metoder.
- Hjalte kigger videre på design af GUI i NetBeans.

Næste møde: tirsdag d. 19. maj.

Absalonlog

Vi har ikke haft problemer med at logge på Absalon i denne uge.

Onsdag, 21. maj

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 11.15-17.00
- Mødeleder: Ditte
- Referent: Hjalte

Tidsforbrug siden sidste gang: Ditte: 6 timer

Anders: $10\frac{1}{2}$ time

Hjalte: $7\frac{1}{2}$ time

Dagsorden:

- Algoritme skal færdiggøres
- Gui-design skal laves
- Hvis muligt skal det kobles sammen

Efter at have kodet på algoritmen er den nu oppe og køre men mangler stadig debug-ging.

Vi har lagt os fast på en design af GUI'en på kodeplan. NetBeans GUI-designer har dog vist sig ikke altid at være ligetil. Specielt kan autogenerated kode skabe problemer.

Fredag, 23. maj

Referatdato: 23. maj

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 11.15-12.30
- Mødeleder: Hjalte
- Referent: Hjalte

Tidsforbrug siden sidste gang:

Ditte: $4\frac{1}{2}$ time

Anders: 1 time

Hjalte: $2\frac{1}{2}$ time

Anders har i øvrigt snakket en lærer, som har svaret på spørgsmål vedrørende vores spil. Vi har aftalt med læreren at vi kan komme ud og teste spillet d. 2. juni på 8 børn i 0.-3. klasse.

Dagsorden:

- Skal ha dannet overblik
- Prioritering af specifikationer

Overblik:

- Vi har en generator der virker men med små bugs. Den virker til både 4x4 og 9x9 men kan optimeres.
- GUI'en er langt fra færdig. GUI-generatoren kan ikke alle de ting vi skal bruge til vores spil, så planen er nu at bruge den autogenererede kode som basis for at hardcode GUI'en.
- Controlleren er påbegyndt men er svær at færdiggøre uden en fungerende GUI.

Prioritering af ikke-baseline specifikationer: Grundet tidspress har vi indset at vi nok ikke når at opfylde alle ikke-baseline kravspecifikationer og har derfor prioriteret dem som følger:

1. Hjælpesystem (opgavebeskrivelsen kræver, at spillet har mulighed for hjælp og vink og bliver derfor klart prioriteret højest. Samtidig er det også det logiske valg, da spillet er til børn.)
2. Feedback (feedback er prioriteret højt, da det har pædagogiske kvaliteter, at brugeren får feedback på rigtige træk, og det kan øge motivationen for at spille.)
3. Pointsystem (et pointsystem kan på samme måde som feedback øge motivationen for at spille, men da det kan skabe stress og fjerne fokus fra indlæringsdelen er det prioriteret lavere, på baggrund af at programmet er beregnet til undervisningsbrug.)
4. Flere sværhedsgrader (er delvist implementeret i form af mulighed for både 4x4- og 9x9-spil. I modellen er det også implementeret at have forskellige sværhedsgrader for plader af samme størrelser, men da spillet er beregnet til børn, har vi i første omgang ikke implementeret de mest avancerede af de logiske metoder, og programmet kan ikke opfylde kravet om hurtig responstid på generering af større plader, før vi implementerer disse.)

5. Mulighed for at spille med symboler i stedet for tal (dette er prioriteret lavest, da vi efter samtale med en skolelærer fik det indtryk, at børnene sagtens ville kunne spille med tal. Derfor vil den ekstra valgmulighed blot forvirre i vores øjne.)

Vi mødes igen søndag kl 9.

Anders og Ditte vil, hvis tiden er til det, prøve at lave noget debugging på generatoren. Vi har ikke oplevet nogen problemer med Absalon

Tirsdag, 27. maj

Referatdato: 27. maj.

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 9.30-11.00
- Referent: Ditte
- Mødeleder: Hjalte

Timer brugt siden sidst:

Anders: 12 timer på at implementere forskellige dele af programmet.

Ditte: 15 timer på at implementere forskellige dele af programmet.

Hjalte: 8 timer på at implementere forskellige dele af programmet.

Dagsorden:

1. Status, herunder gennemgang af kravspecifikation.
2. Diskutere hvordan hjælpesystem og pointsystem skal se ud og hvordan det skal implementeres.
3. Oversigt over hvad vi skal nå i dag.
4. Evt.

Status

1. Muligvis er nogle af sudokuerne (observeret ved 4x4) ikke entydige idet alle felter med mere end et tal er slettet. Dette skal undersøges nærmere. En eventuel løsning er, at sørge for at der er forekomster af alle tal på spillepladen pånør højst ét.

2. Vi har koblet GUI sammen med controller og model, og vi har et kørende program.
3. Feedbacksystemet er næsten færdiggjort.
4. Hjælpesystemet er næsten færdiggjort.
5. Generelt mangler debugging.
6. Koden skal gennemgås, dvs. alt ubrugt og udkommenteret skal fjernes og der skal lave javadoc kommentarer til alle metoder.
7. Vi mangler at gøre programmet mere indbydende og lækrere, dette består bl.a. i at lave ikoner til knapperne, lave baggrundsfarver/-billeder, evt. sætte lyde på (skal kunne slås fra), m.m.
8. Kravspecifikation: Algoritmekrav (1), Lav responstid (3), simpel grafisk brugerflade (2), feedbacksystem (9), hjælpesystem (5) er implementeret på nær nogle småting, som sandsynligvis bliver rettet i dag. Følgende krav mangler: Mulighed for flere sværhedsgrader (7) (vi har det allerede i den forstand at man kan vælge mellem 4x4 og 9x9), pointsystem (4), mulighed for at spille med tal eller symboler (8) (hvis dette skal med).
9. Vi har stillet spørgsmål vedr. programmet og vores ideer til to forskellige folkeskolelærere, som vi vil tage højde for og skrive ind i vores argumentation.
10. Desuden mangler vi testspecifikation, installations- og brugervejledning, beskrivelse af det endelige design, skrive rapporten.

Kommentar til krav nummer 8: mulighed for at spille med tal eller symboler

”Eleverne arbejder med / lærer tallene i bhv.kl., men der er stadig nogle elever, som ikke er sikker i tallene over 5 i begyndelsen af 1.kl., så muligheden for at vælge symboler for tallene vil sandsynligvis gøre det spiseligt for alle. Kan det laves, så eleven selv kan vælge tal eller symboler?”, Marianne, folkeskolelærer

Da vi har besluttet kun at lave symboler i 4x4 sudokuerne (det bliver for uoverskueligt i 9x9), konkluderer vi ud fra ovenstående citat, at det ikke er nødvendigt med symboler.

Hjælpesystem og pointsystem

Hjælpesystemet skal fjerne alle forkerte tal, hvis der er nogen, ellers indsættes et rigtigt. Dette skal vi have feedback på, når vi besøger Mariendal SFO på mandag, da vi ikke er sikre på om børn kan forstå at en enten gør det ene eller det andet.

Pointsystemet skal være en timer, der stopper når sudokuen er løst. Den skal være diskret, så svage elever ikke bliver meget pressede af den. Desuden skal det tage højde for

om brugeren får hjælp, dvs. tillægge ekstra tid og eventuelt tælle op hvor mange gange der bliver givet hjælp. Hvis vi får tid skal en highscoreliste implementeres.

Hvad skal der laves i dag

- Bages kage til øvelsestimerne.
- Tage beslutning om knappen tilbage/ny. Generelt have lagt os fast på indholder af GUI'en samt størrelsen (hvilken størrelse er rimelig).
- Begynde på testspecifikationen (Anders).
- Pointsystemet skal implementeres (Ditte).
- GUI'en gøres mere lækker (Hjalte).

Næste møde: Fredag d. 30. maj.

Fredag, 30. maj

Referatdato: 30. maj

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 14-14.30
- Referent: Ditte

Timer brugt siden sidst:

Anders: 16 $\frac{1}{2}$ heraf 2 timer på endeligt design og resten på GUI.

Ditte: 13 timer på debugging af modellen, endeligt design, implementation af timer m.m..

Hjalte: 16 $\frac{1}{2}$ alle brugt på GUI'en.

Dagsorden:

1. Status
2. Hvad skal der nås i dag
3. Evt.

Status

Spillet kører, nogle småting mangler i GUI'en, der skal skrives rapport, testspecifikation, installations- og brugervejledning, og endeligt design. Desuden mangler der stadig at blive ryddet op i koden.

Vi har nu afgjort at der i vores endelige design ikke skal være mulighed for forskellige sværhedsgrader inden for 4x4 og 9x9, da dette mulgives vil forvirre børnene (især de små) med flere knapper og valgmuligheder. Desuden er der mange børn der aldrig har spillet sudoku, hvilket betyder, at muligheden for at vælge 4x4 og nemme 9x9 er nok.

"jeg spurgte i dag i min 2.klasse om nogen af dem løser sudokuspil. Der var 7 ud af 20, som gør det af og til, både sudoku bag på avisen og på computeren." Marianne, folkeskolelærer.

I stedet vil vi efter vores brugertest på mandag indpasse niveauet på pladerne, så det passer til børnene.

Niveaugraduering er dog allerede implementeret i modellen, så det kunne relativt nemt implementeres i GUI'en.

I dag

I dag skal vi have skrevet designdelen færdig.

Næste møde: Søndag kl. 10.

Absalon

Vi har ikke haft nogen problemer med Absalon i denne uge.

Søndag 1. juni

Referatdato: 1. juni.

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 17.00-18.00
- Referent: Hjalte

Tidforbrug siden sidste gang:

Ditte: 11 timer

Anders: 7 timer

Hjalte: 11 timer

Dagsorden:

- Status
- Uddelegering
- Diskussion af testspecifikation med særligt henblik på brugertests

Status: Brugermanualen er stort set færdiggjort, men mangler stadig en FAQ / hjælp til løsning af problemer.

Vi mangler at lægge sidste hånd på koden - finpudsninger.

Vi mangler at skrive testspecifikationer.

Uddelegering: Anders laver test færdigt og ser på hvordan JAR-filen opfører sig på systemer uden Java Runtime Environment installeret. Han laver desuden en cd med programmet på til brugertesten mandag.

Ditte kigger på brugertestspecifikationer.

Hjalte rydder op i koden.

Testspecifikationer: Vi har valgt ikke at teste samtlige metoder i modellen, da vi laver vores tests ud fra vores kravspecifikationer. Vi tester de ting i modellen, der er vigtige for funktionaliteten, til gengæld både mht. funktionalitet og hastighed.

Diskussion af brugertests: Så vidt det er muligt vil vi udføre vores test således at:

- Pædagogen styrer, hvad der skal ske (kun hjulpet af cd og brugermanual) uden indblanding fra os, da det er sådan, det vil foregå, hvis spillet skal i brug.
- Vi observerer, tager noter og stiller spørgsmål til de enkelte børn, så vi har mest muligt at tage med fra testen.
- Til sidst sætter vi os i en rundkreds med børnene for at høre, hvad de har af kommentarer til spillet.

Vi har ikke oplevet nogen problemer med Absalon.

Mandag 2. juni

- Deltagende: Hjalte, Ditte, Anders
- Tidspunkt: 16.00-17.30
- Referent: Hjalte

Tidsforbrug siden sidste gang:

Ditte: $2\frac{1}{2}$ time

Anders: $2\frac{1}{2}$ time

Hjalte: $2\frac{1}{2}$ time

Efter brugertesten holdt vi igen møde for at diskutere resultaterne af vores test. Testen var generelt en succes, da børnene var glade for at spille spillet.

På baggrund af testen har vi dog bestemt at ændre følgende, inden vi afleverer endelig programkode:

- Vi skal have rettet en bug vedrørende vores toggle buttons, da det i særlige tilfælde er muligt at indsætte tal, uden at en knap er toggled.
- Funktionen af vores vink-knap skal ændres, da den gør for mange ting selv. Dette forvirrer børnene, og en rettelse vil forhåbentlig medføre, at brugerne hurtigere og nemmere bliver klar over funktionen af knappen.
- Nye ikoner på henholdsvis "Ny" og "Vink" knappen, da det ikke i alle tilfælde virkede klart, hvad knappen var til.
- En forøgelse af antallet af tomme felter i vores 4x4-sudoku, både fordi de var alt for nemme, og fordi mængden af feedback i en meget nem sudoku lod til at skabe forvirring om, hvorvidt et tal var rigtigt indsat, når der ikke kom feedback.
- Timeren skal være tydeligere, idet børnene ikke først opdagede den, men generelt var glade for den. Ydermere vil vi tilføje en vink-tæller ved siden af timeren, da børnene gik meget op i, hvor mange gange de havde "snydt" / lagde en ære i ikke at gøre det.

Vi har ikke oplevet nogen problemer med Absalon i denne uge.