Et computerspil til matematikundervisning i folkeskolen

Robotværkstedet

Jeppe Hartmund, S103458

Andreas Slott Jensenius, S093199

27-06-2013

Institut for Matematik og Computer Science

Danmarks Tekniske Universitet

Matematiktorvet

Building 303 B

2800 Kgs. Lyngby

Tlf: 45 25 30 31

Bachelor afhandling projekt nummer 20

Resumé

Denne rapport beskriver et matematikspil, der primært har som funktion at underholde spilleren, men bruger matematikken som et nødvendigt værktøj, der giver mening for spillets historie og atmosfære. Gennem analysen findes frem til de delproblemer der følger gennem funktionelle og ikke funktionelle krav, samt en beskrivelse af de værktøjer og teknologier, vi har valgt at bruge til at løse problemstillingen. Designet beskriver den måde matematikspillet er blevet udtænkt og hvordan de forskellige dele af programmet relaterer til hinanden. De mest interessante dele af implementationen bliver beskrevet i implementationsafsnittet. Til projektet er der knyttet en undersøgelse af det endelige produkt, hvor spillet er blevet testet på tre folkeskoleklasser, og resultatet af dette, samt udtalelser fra den, til projektet, tilknyttede lærer, hvis indsamlede data vil blive redegjort for og behandlet i resultatafsnittet og diskuteret i diskussionsafsnittet. I konklusionen vil vi forsøge at besvare om problemerne, der blev redegjort for i introduktionen, er blevet løst.

Tak til

Vi vil gerne sige tusind tak til Erik Ottar Jensen, for det første, for et godt samarbejde og sparring omkring spillets funktionalitet og mulige retninger projektet kunne tage, og for det andet for at afsætte tid i hans arbejdsdag til at vi kunne få lov til at afprøve vores spil.

Mange tak til vores vejledere Jeppe Revall Frisvad og Niels Jørgen Christensen for at have udbudt projektet og tilrettelagt det.

Og endnu en tak til Jeppe Revall Frisvad for hans hjælp og vejledning på de møder vi har haft sammen om spillets udvikling og design.

Vi vil også gerne sige mange tak til Christian Kaiser, der har været en uvurderlig hjælp, ved at have leveret størstedelen af billederne til programmet.

Der skal også lyde en tak Christine Thue Poulsen for at have testet spillet undervejs i forløbet og kommet med løbende feedback.

Indholdsfortegnelse

1. Introduktion

1.1 Indledning

Matematik kan være svært for mange at lære, men der er nogle områder af matematikken der skal læres meget tidligt for at danne grundlag for elevens forståelse af matematik. Et eksempel på dette er positionssystemet, det at et tal kan deles op i ettere, tiere, hundreder etc. Hvis dette læres tidligt kan det være en kæmpe hjælp for forståelsen af mere avancerede emner inden for matematik.

Udenfor skolen bruger eleverne meget tid og energi på at løse komplekse problemer i diverse computerspil. Ved at kombinere læring og computerspil er det tanken at man kan få brugt den energi og entusiasme eleverne har omkring computerspil til at gøre dem bedre til, i vores eksempel, matematik.

Rapporten her vil gennemgå den designprocess og implementation, der skal munde ud i et matematikspil, som primært er sjovt, og hvor det at lære matematik stadig findes men ikke er tydeligt for eleven. Efter introduktionen der beskriver problemet vil rapporten gennemgå analysen af problemstillingen, hvor den vil beskrive kravene til produktet, hvilke funktioner det kræver af værktøjerne og hvilke værktøjer der kan udføre dette. Designafsnittet forsøger dernæst at finde en løsning på de delproblemer analysen fandt frem til. I implementationsafsnittet beskrives nogle af de mere interessante implementationer. Ved siden af rapporten og produktet blev der foretage en undersøgelse af produktet, hvis resultater udgør rapportens resultatafsnit. Efter dette vil diskussionsafsnittet diskutere de resultater vi har fået i resultatafsnittet. Til sidst følger en refleksion over projektforløbet og en konklusion på rapporten.

1.2 Problemstilling

Matematik danner grundsten for de tekniske og naturvidenskabelig fag, og er generelt et meget vigtigt emne for alle i samfundet. Det er derfor problematisk, hvis en stor andel af danske unge har store vanskeligheder med matematikken. Avisen Ingeniøren skriver (baseret på pita undersøgelsen):

" Andelen af de dårligt præsterende elever i Danmark [...] ligger støt på omkring 16 procent. [...]

Eleverne i bunden kan kun svare på spørgsmål med et familiært indhold og kan kun udføre rutineprocedurer efter direkte instruktioner. De allersvageste kan ikke lave de mest basale matematikopgaver." [1]

Problemet kan begynde allerede i de helt små klasser, hvor det så kan forgrene sig. Hvis man ikke forstår de grundlæggende matematiske begreber bliver matematikken man lærer i senere klasser meget svær at forstå. En analogi for dette er at bygge et hus. Du lægger først fundamentet, dernæst rejser du murene og til sidst lægger du tag på. Hvis eleverne allerede falder bagud når fundamentet lægges, kan de ikke ordentligt rejse deres mure, og når de kommer til at skulle lægge tag på vil de for alvor få problemer. Det er derfor vigtigt at få en grundig forståelse af de grundlæggende matematiske begreber, helt fra de små klasser.

[1] Reference 1

Hvordan får man så eleverne i de små klasser interesseret og engageret i matematik, og fremmer deres forståelse? Tavleundervisning har traditionelt været vejen frem, men når man har mere lyst til at løbe rundt og lege eller spille computer så er det svært at sidde stille og koncentrere sig om tavlen.

Et andet problem er hvis alle matematikopgaver har en bestemt, fast opstilling. Erik Ottar Jensen som er under uddannelse til folkeskole matematiklærer og som har været tilknyttet projektet fortalte om nogle tests han havde lavet i sin klasse. Ved to ens plus regnestykker, opstillet på hver deres måde, var antallet som regnede forkert langt større for den af de opstillinger som de ikke var vant til[2].

Ved projektets start blev vi desuden præsenteret for nogle områder i matematikken som eleverne ofte havde problemer med. Et af dem var positionssystemet, altså det at forstå sammenhængen mellem enere, tiere, hundreder osv. Det er et ret centralt emne og er et problem vi gerne vil arbejde med.

Et problem andre matematik computerspil har, er at det er svært for læreren at se om eleverne får noget ud af det, og hvad de potentielt har problemer med.

Det sidste interessante problem er differentiering af undervisningen. Som man også kan læse i den førnævnte artikel fra Ingeniøren er antallet af top præsterende faldet. Så kan man finde en måde at hjælpe de svageste til at blive bedre, og samtidig også udfordre de dygtigste?

Målet for dette projekt er at give en løsning på disse problemer ved hjælp af et computerspil. Men hvorfor et computerspil og hvordan kan vi løse problemerne med det?

Computerspil er interaktive og kræver brugerens aktive deltagelse, og det er et medie som børn er vant til at bruge og som de forbinder med sjov og underholdning. Dette skulle gerne hjælpe på det førstnævnte problem, hvis vi altså formår at lave et interessant spil. Vi mener desuden at det er vigtigt at eleverne ikke får indtryk af at de sidder og laver matematik opgaver, men at matematikken blot er et værktøj for dem. Dette vil vi fortælle mere om i afsnit **2.1 Spilkoncept**.

Computerspil giver os desuden muligheden for at pakke matematikken ind på en ny måde, hvor de vante metoder ikke blindt kan bruges, men de underliggende begreber skal forstå for at kunne bruge dem. Vi mener derfor ikke det nytter noget at have et spil, som vil have dig til at udfylde 3 + ____ = 5 i en tekst boks, da man så bare har flyttet tavleundervisningen hen på en computerskærm, men man derimod skal finde en alternativ måde at præsentere matematikken på.

For at sikre os at vores spil kan hjælpe eleverne med at lære positionssystemet, skal vi sørge for at det supporterer nogle ikke-standard regne operationer som manipulerer med tallene på en måde som tvinger eleverne til at tænke på tallene på den rigtige måde.

Til at give en lærer den feedback om sine elever han ønsker, vil vi designe en Lærer Klient som kan vise information om den enkelte elev, såsom hvor hurtigt han klarer en given bane-type, såsom en bane-type som fokuserer på gange og dividere, og plotte det i en graf så man også kan se om han forbedres ved at spille spillet. Derudover vil vi gøre så man kan se hvilke regneoperationer, som en elev foretrækker, og

[2] Bilag 1

hvilke eleven ikke bruger så ofte. Dette kan sige noget om, hvor tryg han er ved dem, og hvor godt han forstår dem.

Det sidste problem kan vi løse ved at designe vores computerspil med stigende sværhedsgrader, så det kan udfordre alle, og vi kan sørge for at det ikke har for meget linæritet, som kan lukke det af for de svageste, eller kede de dygtigste.

1.3 Relaterede arbejde

Af relaterede produkter vi fokuserer på, er de alle sammen computerspil. Man kunne have valgt at beskrive legetøj eller eksisterende undervisningsmateriale i form af bøger eller film, men idet problemet består i at beskæftige sig med computerspil som indlæringsværktøj, vil vi primært være i stand til at drage konstruktive paralleller til andre computerspil, og idet computerspil som legetøj efterhånden er ligeså normalt som de mere klassiske former for legetøj, er der derfor tilstrækkelig med eksempler på disse.

1.3.1 Populære computerspil

1.3.1.1 Minecraft

Minecraft er et spil der lægger vægt på spillerens kreativitet og giver spilleren mere eller mindre frie tøjler til at ændre på den virkelighed som genereres ved spillets start. Dette overskueliggøres ved at bryde verden op i uniforme klodser, der kan sammensættes til potentielt at ligne alt. En stor kvalitet ligger i at spilleren sætter sine egne mål og der er ikke nogen klart defineret slutning på spillet. Spilleren laver sin egen dagsorden for hvordan spillet skal opleves og på diverse fora og mellem venner viser de deres kreationer frem. I samme boldgade findes der mange Minecraft spillere der har haft succes på Youtube med at optage og vise dem selv spille.

Fordi spillet rummer så mange muligheder for at være kreativ og nytænkende, samt det faktum at det stadig er under udvikling og løbende får tilføjet nyt indhold er der også spillere på Youtube der laver videoer, hvor de fremlægger nye features og hvordan disse kan bruges i spillet.

Det kan altså betragtes som en moderne form for Lego, idet du skaber dine egne projekter ud fra en idé og kan sidde og pusle med den indtil du er tilfreds.

Det at Minecraft tillader brugeren at sætte sine egne mål giver en kæmpe frihed i spillet, der snarere føles som en legeplads end et computerspil, og er noget der kunne videreføres til vores matematikspil.

1.3.1.2 The Sims

The Sims er et andet spil der fokuserer på kreativitet. Spillet handler om en familie, som du selv skaber og er en form for simulation af deres liv. Spilleren bygger huset, sørger for familiens behov og hverdagsrytme og administrerer deres økonomi. Ligesom Minecraft er der ikke noget mål med spillet udover de du selv sætter. Ligesom Minecraft kunne sidestilles med Lego, kan man her drage paralleller til at lege med dukker i et dukkehus. Forskellen er selvfølgelig at med The Sims følger en følelse af at have udrettet noget. En familie du skaber kan samle en opsparing, få børn, udvide huset, flytte til et større hus osv.

Familiemedlemmerne bliver ældre og dør, børnene bliver voksne og får selv børn. Der er altså hele tiden noget nyt at give sig til. Dette gør at spillet beholder sin værdi i meget længere tid end klassiske spil, som Nintendos Super Mario, der har en lineær historie.

Det der kan videreføres til vores produkt er det, at spillet tillader dig at fortsætte så længe du har lyst.

1.3.2 Eksisterende matematikspil

1.3.2.1 Manga high – A tangled web

Dette spil er et lille matematikspil der introducerer begrebet om vinkler og skal lære om at tælle til hhv. 90, 180 og 360, men dette er ikke hovedfeaturen i spillet. Spillet handler om en robotedderkop, der skal bevæge sig gennem en række små baner af stigende kompleksitet der finder sted i midten af et tandhjul. Spilleren styrer spillet med piletasterne, der roterer banen i hver deres retning. Når banen bliver tiltet, flytter figuren sig. Med figuren skal spilleren samle nogle mekaniske fluer, hvorved en port åbner sig til næste bane. Matematikdelen bliver introduceret i 3. bane, hvor der er streger på tværs af banen. Stregerne fungerer som mure, der kan åbnes ved at skrive linjens vinkel.

Problemet dette spil er at matematikdelen er meget abstrakt i forhold til spillets andre dele. Alle elementerne har deres egen berettigelse og virker integreret i spillets miljø og oven på dette vises nogle linjer der af en eller anden grund forhindrer robotedderkoppen i at komme videre. Det er vigtigt at hvis man har besluttet sig for et tema, så også at følge det for at det ikke skal se halvhjertet ud.

Det er vigtigt, når vi designer vores spil ikke at lægge en uforklarlig spilmekanik ovenpå, hvis ikke det giver mening i spillets historie, bare for at gøre et almindeligt computerspil til et matematikspil.

1.3.2.2 Manga high – Transtar

Spillet her er ligesom det forrige fra siden Manga high og beskæftiger sig også med geometri. Spillet foregår ude i rummet, formet som et koordinatsystem, hvor ”rumskibet” TranStar skal dirigeres gennem en såkaldt StarGate. Rumskibet har en simpel asymmetrisk form og porten har en tilsvarende form, denne er blot roteret eller spejlet. Målet er nu, ved hjælp af linjer, der kan spejle rumskibet og ormehuller, der kan rotere, at få figuren til at passe i hullet ved så få træk som muligt. Banerne bliver mere og mere komplekse og der introduceres fjendtlige rumskibe du skal undgå og andre forhindringer.

Hvis man træder et skridt tilbage og kigger på spillet er det i virkeligheden geometriopgaver, som man kunne forestille sig at støde på i folkeskolen, leveret på en måde, således at det føles som leg. At man ikke tager det ud af kontekst fra koordinatsystemet, viser også at stregerne i virkeligheden er en kæmpe hjælp, når man skal løse denne her type problem, og det bliver hurtigt meget nemt at se hvad man skal gøre i de fleste baner. Den måde spillet integrerer rigtig matematik, uden at det føles presset ned over hovedet, gør spillet meget mere værdifuldt for elevens tilegnelse af matematiske færdigheder, han senere hen kan bruge, hvilket er noget der bestemt kan videreføres i vores matematikspil.

1.3.3 Opsummering

Det der gjorde de populære computerspil gode og giver spilleren lyst til at spille videre bliver ikke videreført til matematikspillene. Matematikspillene har en historie, spændende detaljeret grafik og pointsystemer der angiver, hvorledes du klarede banen.

Minecraft og The Sims har fanskarer på flere millioner af mennesker, men har hverken en historie, et pointsystem og Minecraft's grafik består af teksturer på 16 x 16 pixels. Til gengæld udfordrer de spillerens kreativitet og giver uanede muligheder for hvordan en opgave kan løses. De er åbne og har ikke en fast defineret slutning.

Da de versioner af matematikspillene vi kunne teste har været gratis prøveversioner har det ikke været muligt at undersøge hvordan spillet slutter, men de lægger ikke op til en åben slutning.

2 Metode

I dette afsnit vil vi gennemgå spilkonceptet som vi har arbejdet med, og hvordan vi kom frem til det. Spilkoncept afsnittet vil også introducere mange af de begreber vi senere vil bruge. Dernæst vil vi gennemgå de forskellige krav vi har til projektet, samt de teknologier vi vil få brug for.

Til sidst vil se på de design beslutninger vi har arbejdet med og taget projektet igennem. Dette vil blive gjort på et abstraktionsniveau hvor vi udelader mange tekniske detaljer eller dybere forklaringer af konkrete løsninger, da dette vil blive gennemgået i afsnit 3 Implementation.

2.1 Spilkoncept

Af de populære spil allerede omtalt, Minecraft og The Sims, er det klart at fællesnævneren er kreativitet og leg. I stedet for at presse spilleren gennem en foruddefineret lineær historie overlades spilleren med en række værktøjer og uendelig mange muligheder. Det er klart at et spil der tillader dette er et stort og potentielt dyrt projekt, men kernen i det kan sagtens bruges i et mindre matematikspil. Matematik deler nogen af disse værdier og tillader uendelige muligheder med de værktøjer den introducerer.

I vores spil skal spilleren samle et kredsløb af ledninger og komponenter for at ændre et antal inputtal til et antal ønskede outputtal. Ideen er at spillet skal opfordre spilleren til at lege med tallene og afprøve forskellige komponenter og forhåbentligt lære, dels at matematikken er uafhængig af form, dvs. et regnestykke er ikke afhængigt af den måde det er skrevet op, men også det problem som vi, som udgangspunkt, valgte at fokusere på hos folkeskoleeleverne, netop positionssystemet.

Siden vi begge har interesse inden for computere og teknologi har vi valgt at spillet skulle låne sit udseende og præmis fra elektronikken. Ideen med at tage et tal og ændre det ved hjælp af nogle forskellige operationer ligner også på mange måder en printplade med mikrochips og andre elektroniske komponenter. Dette overvejede vi sammen med ideen om at have en fabrik der samlede et produkt, hvor løsdelene var tal der kørte ad et transportbånd. Den store forskel på disse to løsningsmuligheder er dog ikke kun kosmetisk.

I begge løsninger har brugeren en printplade som de placere komponenter, som udføre regneoperationer, på og forbinder dem. Uden transportbånd har brugeren nogle statiske tal i starten som de kan bruge, hvorimod med et transportbånd ville tallene ændre sig over tid som nye tal-pakker ankom på båndet. Dette giver mulighed for at komponenternes outputværdi ændres over tid. Den anden måde, som er den måde den endelig printplade er tænkt foregår strømmen af værdier uafhængigt af tid, dvs. start tallene ændre sig ikke. Det tager altså ikke noget tid for værdien at bevæge sig gennem systemet, hvilket giver mere ro for spilleren at koncentrere sig om det bagvedliggende regnestykke.

Spillet introducerer et spilbræt formet som en printplade. Spilbrættet har et ”koordinatsystem” af punkter hvortil de forskellige dele skal sammensættes. I højre side vises komponenterne i en værktøjskasse eller på et bånd eller lignende. I toppen og bunden af spilbrættet vises hhv. inputs og outputs, tildelt et tal så det tydeligt fremgår hvilke tal der hhv. er til rådighed eller ønskes. På spilbrættet findes afhængigt af banen et antal fikserede komponenter, der ikke nødvendigvis skal bruges, men bidrager til antallet af komponenter ved udregning af dine point. Det er nu spillerens opgave at lægge komponenter på spilbrættet og forbinde

dem til spilbrættets inputs og outputs. Når alle outputs modtager det ønskede tal udregnes point og spillet er slut.

Idet vi tager udgangspunkt i positionssystemet, hvorved der menes at tal kan opfattes som kombinationer af ettere, tiere, hundreder osv., har vi valgt at tage nogle komponenter med der ikke er gængse regneoperationer. Vi har tre filterkomponenter der hver især filtrerer hhv. ettere, tiere og hundreder fra et tal og sender disse ud af et output og resten ud af et andet. En feature som vi oprindeligt ville have med var en farvekode af positionssystemet således at børnene kunne opfatte et tal, fx 123, som en rød ener, en blå toer og en grøn treer. Problemet med det er at det kan være svært at forudse om dette vil hjælpe til forståelsen af positionssystemet eller i stedet være så anderledes at det kan være svært for spilleren at drage paralleller til det senere hen. Der er endda en risiko for at det kan bidrage til forvirringen af et emne, som de i forvejen har problemer med at lære. Et andet problem ved det er håndteringen af tal på over 3 cifre. Hvis spillet skal være et spil hvor spilleren skal udforske og lege uden at få ført hånden ville vi enten skulle sikre os at tallene ikke overstiger 3 cifre, hvilket måske ville gå ud over spiloplevelsen, eller også lave en automatisk måde at tildele højere talpositioner en farve. En anden måde at inddrage farvekoden var at lade farverne være bundet til tallene og ikke positionerne. Dette ville kræve at vi satte os ned og besluttede hvad regneoperationer med forskellige farver skulle farve outputtet. Dette kunne potentielt lede til endnu mere forvirring, da det kan være svært at forudse hvad der ville ske med outputtet. Til gengæld kunne man inkludere farvede filtre der fx filtrerer alle røde tal fra og selvfølgelig beholder deres positioner, hvilket ville kræve at eleverne brugte meget energi på at tænke over tallenes positioner. Systemet ville dog bidrage meget til spillets kompleksitet og kunne meget tænkeligt gøre det svært for spilleren at komme rigtig i gang.

Den oprindelige idé med spillet inkluderede at størrelsen på spilbrættet skulle forøges i takt med sværhedsgraden. Intuitionen siger at med et mindre spilbræt er der færre muligheder for placering af komponenter og derfor også større sandsynlighed for at man finder den rigtige løsning hurtigere. Problemet med dette er at en mindre brætstørrelse for det første begrænser antallet af mulige løsninger, men også begrænser spillerens overblik, da alting bliver forstørret op. Derfor valgte vi at beholde samme brætstørrelse gennem hele spillet og på andre måder justere sværhedsgraden.

Spillet tillader at man lader outputs fra komponenter stå ubrugte. Dette giver langt flere muligheder for at lave baner, hvor det ikke er muligt at regne ud hvor mange komponenter du skal bruge af hver type for at antallet af indgange skal passe til antallet af udgange. Det giver også spilleren langt flere muligheder for at finde på smarte løsninger, der kræver færre komponenter end vi har forudset.

Fikserede komponenter, komponenter der fra banens start sidder på brættet og ikke kan flyttes, kan bruges til at presse eleverne til at afprøve komponenter som de måske ellers ikke ville bruge. Idet banerne meget gerne skulle kunne løses på mange forskellige måder, er der en risiko for at spilleren forsøger at undgå komponenter, hvis matematiske funktion eller spilmekaniske funktion ikke falder spilleren lige for, men i det mindste bliver de opfordret til at bruge dem, da en løsning kan være designet til at drage nyttet af netop den funktion som det fikserede komponent tilbyder.

Dynamisk genererede baner er en feature, som vi, da vi kom på den, ville have skulle fylde meget mere end vi endte med at lade det. Dynamisk genererede baner har mange kvaliteter der taler for, men det er på bekostning af andre væsentlige kvaliteter. Grunden til at det kom på tale, var at spillet ville have en enorm genspilbarhed, idet hver bane er forskellig (i praksis i hvert fald). Desuden ville vi ikke skulle designe

banerne enkeltvis, hvilket potentielt kunne spare os tid. Problemet er at sikre at sværhedsgraden stiger i det rigtige tempo. Kompromiset er at spillet kommer med et antal predefinerede baner der er designet således at spilleren bliver ført igennem alle spillets features i stigende sværhedsgrad. Ved siden af disse baner kan spilleren også vælge en dynamisk genereret bane, der ikke tæller med i spillets overordnede forløb.

Tegnestilen er noget vi diskuterede meget tidligt I processen og er vigtig for hvordan målgruppen oplever spillet. Vi har valgt at have en stil der ser meget håndtegnet og tegneserieagtig ud, hvilket passer tilstrækkeligt til spillet og målgruppen, og samtidig undgår vi paint-agtig grafik og behøves ikke at forsøge at lave noget som er virkelighedsnært.

Som incitament til at spille spillet har vi valgt at historien skulle finde sted i et værksted/laboratorium, hvor spilleren står over for deres ødelagte robot ven. Ved at løse banerne, der knytter sig til hver af dens ødelagte legemsdele, repareres robotten og til sidst når en gren af baner er løst, vil den tilknyttede legemsdel fremstå repareret.

2.2 Analyse

2.2.1 Funktionelle krav

I de to følgende afsnit vil vi gennemgå de funktionelle krav, som vi har fra vores koncept, og fra læreren som spillet designes til - Erik Ottar Jensen, og de overvejelser som vi har gjort os om de opstillede krav, og mulige løsninger til dem.

2.2.1.1 Opstilling af krav

De funktionelle krav vi selv stillede til vores spil og den platform vi ville udvikle det i:

1. Tegne figurer på skærmen
2. Afspille lyd
3. Vise billeder
4. Drag-and-drop af komponenter
5. Simple animationer

De funktionelle krav som vi har fået stillet af vores vejledere og Erik Ottar Jensen:

6. Indsamle og opbevare data
7. Have brugerprofiler så data kan sorteres for den enkelte elev
8. Lave grafer, eller anden repræsentation af data
9. Køres i en browser og tilgås fra internettet
10. Være kompatibelt med de mest almindelige browsers

2.2.1.2 Hvordan opfylder vi kravene

For at sikre os at spillet og den platform, vi vælger, opfylder kravene fra forrige afsnit, går vi her lidt mere ind i de tekniske detaljer for, hvad der skal til for at opfylde kravene.

Krav 1 til 3 er trivielle i moderne sprog og platforme, og vi vil derfor ikke gå i dybden med dem.

Krav 4 er også triviel, men der findes to primære måder at håndtere drag-and-drop på. Enten kræver det, at vi har adgang til mouse events, såsom MouseButtonDown og MouseMove, eller også kan der bruges, som en indbygget funktionalitet i sproget/platformen. Indbygget funktionalitet er at foretrække, men mouse events kan også bruges uden de store problemer.

Krav 5 kan ligesom krav 4 opfyldes ved enten indbygget support for det, eller ved programmatisk at rykke rundt eller ændre på et givet objekt. Hvis animationer skal bruges, ville det helt klart være at foretrække, hvis der var indbygget support for det, da det er noget mere kompliceret at lave fra bunden.

Krav 6 og 7 kræver en form for database, hvilket der findes mange af, og vi kan vælge typen uafhængigt af sprog og platform, så det stiller ikke skrappe krav.

Krav 8 kan som tidligere krav enten opfyldes ved indbygget support, eller ved at håndlave det. Da krav 1, at tegne på skærmen, er triviel er det også trivielt at repræsentere data grafisk. Indbygget funktionalitet ville dog klart være at foretrække, da det ville give os mange flere muligheder, for at bruge dataen, uden væsentlig arbejde fra vores side.

Krav 9 og 10 hænger sammen, og er det, som primært indskrænker vores muligheder når vi skal vælge sprog og platform. Så inden vi lægger os fast på et sprog eller platform, skal vi sikre os at det har mulighederne for at blive kørt i en browser, og kan afspilles fra en server.

2.2.2 Ikke-funktionelle krav

I følgende afsnit opstilles de mest relevante ikke-funktionelle krav til matematikspillet, hvormed der menes krav til programmet, der ikke omhandler en funktion programmet skal udføre men en kvalitet ved programmet som helhed.

2.2.2.1 Tilgængelighed

Med tilgængelighed menes, hvorvidt produktet kan nå ud til så mange brugere som muligt. Skal programmet køres i en browser med et frit tilgængeligt plugin, i en browser helt uden plugins, eller skal der hentes et lukket system som programmet påkræver. Idet programmet skal køres af skoler der kan være underlagt restriktioner fra den ene eller den anden side, skal programmet være mere eller mindre tilgængeligt på de systemer de i forvejen er vant til at benytte.

2.2.2.2 Følelsesfaktor

Et af de vigtigste ikke-funktionelle krav til programmet er at det skal være sjovt at spille, men det er også en af de sværeste krav at sikre sig er opfyldt. Mange forskellige faktorer har indflydelse på, hvor sjovt et spil er at spille. Spillet skal have en veldefineret atmosfære. Grafisk udtryk og lyde kan bidrage til atmosfæren og bestemmer om et spil er uhyggeligt, sjovt eller afslappende. En anden vigtig faktor er at sværhedsgraden er balanceret. Et spil der generelt er nemt og ikke udfordrer spilleren kan hurtigt virke kedeligt og uinspirerende. Hvis et spil derimod er meget svært kan det være frustrerende for spilleren. Spillets historie kan også spille ind, men er ikke nødvendigt. Et spil der fortæller en god historie kan opfordre spilleren til at fortsætte for at følge med i hvad der sker. På den anden side har spil som Minecraft og The Sims oplevet stor succes på trods af at der ikke er en egentlig historie forbundet med de spil. Det gode ved de spil har derimod været spilmekanikken, der har fungeret godt.

2.2.2.3 Platform kompatibilitet

Programmet er kompatibelt med alle systemer, der kan køre Silverlight i en browser, hvilket der alle de store browsere kan på Windows, XP og frem, og Mac med intel processor. Problemet med vores specifikke målgruppe, de små klasser i folkeskolen, er at kommunale folkeskoler ikke altid selv bestemmer, hvilke programmer de må installere på deres computere.

2.2.2.4 Brugervenlighed

Spillet udvikles til en målgruppe der ikke nødvendigvis har den store erfaring med computere og databehandling. Det sætter nogle begrænsninger for hvordan spillet bør styres. Sådan noget som genvejstaster på tastaturet er intuitiv for erfarne brugere da de er vant til at bruge ctrl+z til at fortryde en handling og ctrl+c for at kopiere noget, bare for at nævne nogle få eksempler. Langt mere intuitivt er brug

af musen og knapper på skærmen. Problemet med denne løsning er at det generelt er en langsommere måde at styre spillet på, men overfor målgruppen er det tilstrækkeligt for at sikre dem brugervenlighed.

Andre aspekter af brugervenlighed er at strukturen af spilmenuer ikke skal være for komplicerede, samt knapper skal se ud som om man kan trykke på dem.

2.2.3 Teknologier

2.2.3.1 C#

Vi har valgt C# som det sprog vi vil skrive vores spil i af forskellige grunde. For det første er det et sprog som vi begge kan lide, er gode til og har meget erfaring i. Dernæst er det et stærkt, highlevel objekt-orienteret programmerings sprog, hvilket vi har brug for. Vores eneste bekymring ved at vælge C# er, at det ikke er lige så nemt, som for eksempel Java, at køre fra en webbrowser.

2.2.3.2 Silverlight

Silverlight er et gratis webbrowser plug-in, som kan præsentere medieapplikationer, og give os, som udviklere, et framework vi kan arbejde i. Silverlight supporter dele af .NET Framework, og tillader os at lave vores spil i C# og samtidig køre det i en webbrowser. Ved at give os adgang til .NET Framework, får vi også en masse værktøjer og funktionalitet stillet til rådighed, som er en stor hjælp til udvikling af et software produkt. Silverlight opfylder desuden krav 1-5 samt 9 og 10 fra de funktionelle krav. Drag-and-drop er ikke indbygget, men kan nemt laves med mouse events, men resten har indbygget support. Krav 8 opfylder Silverlight ikke i forhold til indbygget support, men simple grafer kan vi dog selv lave, selvom det vil betyde at vi er begrænset til en enkel type graf, da vi ikke har tid til at lave vores eget graf library til at kunne lave mere end en slags graf.

2.2.3.3 MVVM

MVVM (Model View ViewModel) er et design pattern som bygger på MVC, og er designet specifikt til at lave UIs med. Det er udviklet af Microsoft til Windows Presentation Foundation (WPF) som Silverlight er stærkt inspireret af, hvilket har gjort det til et oplagt valg at følge. [3] Modellen og View'et er omtrent det samme som i MVC, men det som skiller MVVM ud er View-Model, som er en ren koderepræsentation af View'et, også kaldet Code Behind, som derudover også opfylder mange af de samme funktioner som Controller'en fra MVC gør.

Det som gør MVVM ideelt til WPF og Silverlight er data bindings. Databindings er en nem og effektiv måde at lave en løs kobling mellem noget i View'et og i ViewModel'en, så vi kan lave en automatisk, event-drevet måde at synkronisere dem på, som helt lader os undgå at skrive specifik kode til at håndtere det.

[3] WPF Apps With The Model-View-ViewModel Design Pattern, Ref 2

2.2.3.4 XAML

XAML står for Extensible Application Markup Language og er et deklarativt sprog baseret på XML. Det blev oprindeligt udviklet til WPF og er senere blev adopteret af Silverlight, ligesom MVVM. XAML har givet os en nem måde at lave vores brugerflade i, og kan ved hjælp af data bindings, knyttes sammen med resten af vores kode.

2.2.3.5 PHP

PHP, der står for PHP: Hypertext Preprocessor, er et almindeligt scriptsprog til at lave dynamiske hjemmesider. Scriptet bliver afviklet på serveren, hvilket gør det ideelt til at foretage databaseforespørgsler og omdanne eventuel hentet data til html så det kan læses af klienten.

2.2.3.6 MySQL

MySQL, en sammentrækning af navnet My og SQL, der står for structured query language, er et såkaldt relationel database management system og bruges til at holde styr på databaser. De database operationer vores projekt kræver er meget simple, og stiller ikke de store krav til database management systemet. Oprindeligt havde vi tænkt os bare at bruge en Microsoft Access database, men da vi har en MySQL database til rådighed er der ikke nogen fordel i at bruge en anden.

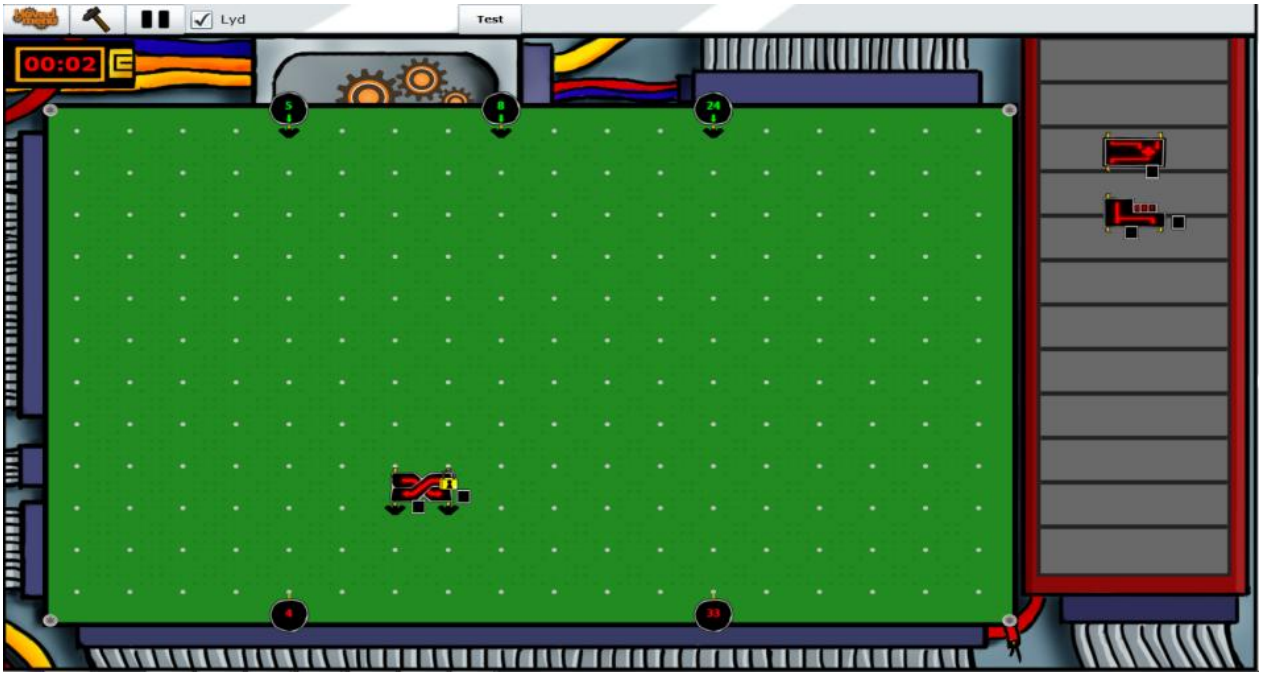
2.3 Design

2.3.1 GUI

Det grafiske brugerinterface, eller GUI, bruges til at præsentere spillet på en brugervenlig måde og give den rette indlevelse for brugeren. I det følgende afsnit præsenteres brugerfladen af de forskellige dele af spillet ud fra disse to kvaliteter.

2.3.1.1 Spilbrættet

Spilbrættet er designet til at ligne en printplade og udseendet på komponenterne er inspireret af mikrochips. Et eksempel på en bane kan ses på Figur 1. For at bevare temaet om at du reparerer en robot og at det er et kompliceret stykke arbejde har vi valgt at stilen skulle være meget teknisk med ledninger og tandhjul som en del af baggrunden og et transportbånd der transportere løsdele ind fra højre. Oprindeligt var det tænkt at komponenterne skulle ligge i en værktøjskasse, hvilket forklarer hvorfor transportbåndet nogen gange vil være omtalt sådan. Grunden til at vi gik væk fra dette var at en værktøjskasse set ovenfra og ned blot er en firkant hvilket var for kedeligt at se på, samt at vi gerne ville have noget blikfang og bevægelse i spillet.



Figur 1 Eksempel på en bane i spillet

Tandhjulene der drejer rundt og den kappede ledning med elektricitet der står ud i blå lyn skal gøre spilbrættet mere levende. Problemet ved et spil som vores, hvor der ikke er meget der bevæger sig, er at spillet kommer til at se stift og livløst ud. Når der sker en smule i baggrunden gør det spilverdenen langt mere levende at se på.

Spilbrættet indeholder et ur, der er tænkt som motivation for spilleren og indikation til læreren om hvorvidt eleven har prøvet at løse problemet selv før han spørger om hjælp. Et problem ved uret kan være at det kan stresse en elev der har svært ved en bane, så meget at det kan frustrere eleven og hæmme kreativiteten. En anden mulig løsning kunne være at skjule uret, men derved mister eleven også muligheden for at evaluere sin egen præstation.

Komponenterne flyttes med musen og det foregår ved simpel drag-and-drop. På samme måde kan en komponentforbindelse trækkes hen over spilbrættet og når der slippes laves der komponentforbindelser hele vejen hen til punktet, såfremt vejen ikke er blokeret. Komponentforbindelserne slettes ved, først at klikke på hammer ikonet oppe i venstre hjørne, hvorved cursoren skifter til en hammer, og dernæst klikke på komponentforbindelsen. En anden måde hvorpå det kunne gøres var at holde fx ctrl nede mens man klikkede på komponentforbindelse, men problemet med det er at vi beskæftiger os med en målgruppe der ikke har så meget erfaring inden for computere og dels er det svært grafisk at vise at denne funktion eksistere, så man risikere nemt at brugeren simpelthen ikke ved den mulighed eksistere da vores målgruppe ikke ville gå ind og læse en manual eller lign.

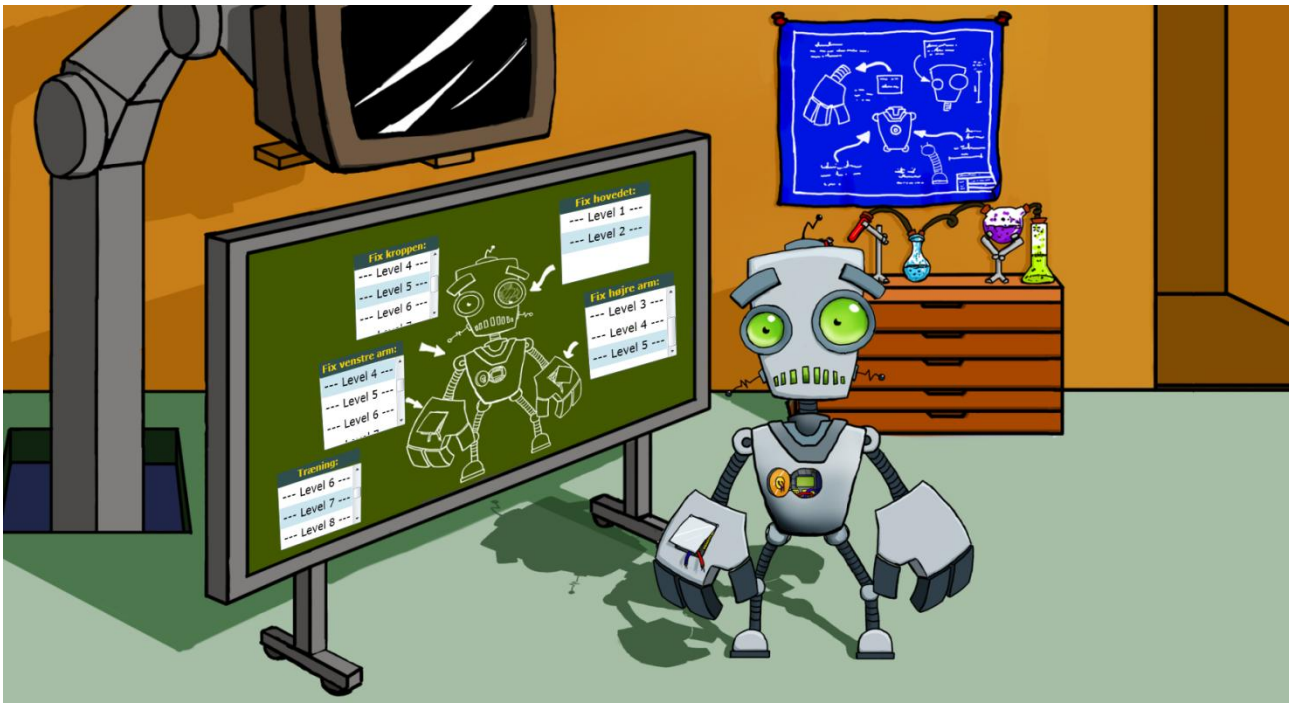
2.3.1.2 Hovedmenu

Hovedmenuen er udformet så det giver mest mulig kontinuitet i programmets stil. Baggrunden er tegnet som printpladen fra selve spillet og overskriften ligner et komponent. Dette er vigtigt for at man stadig skal fornemme at man er inde i spillet når man havner der efter at have klaret en bane. Knapperne er store og tydelige og teksten er klar og læselig. Spillets hovedperson, robotten som du skal reparere, siger velkommen med en tilfældig sætning ud fra en række predefinerede, der bruger elevens brugernavn. Desuden er der et par roterende tandhjul der skaber liv og bevægelse i billedet. På Figur 2 ses hovedmenuen.

Figur 2 Hovedmenuen i spillet, som en lærer med brugernavn Jeppe ville se den

2.3.1.3 Valg af bane UI

Denne menu er den vigtigste menu for historien af spillet, da man her ser laboratoriet som spillet foregår i, hvilket kan ses på Figur 3. Her ses den ødelagte robot og en masse baggrundsdetaljer, der bidrager til den teknologiske og videnskabelige atmosfære. En tavle fungerer som menu, hvor en kridttegning af robotten samt pile fra listbox'ene viser den sektion, hver bane skal reparere. For at vælge en bane dobbeltklikkes på en af felterne i de førnævnte sektioner. Låste baner vil blive vist i gråt for at indikere at man ikke kan gå ind i dem, som er en standard mange programmer anvender sig af. En anden del som er vigtig for historien i denne menu er at når en bane vælges zoomer man ind på det sted på robotten man har valgt, f.eks. kroppen, og man kan så den grønne plade som selve spillet foregår på.

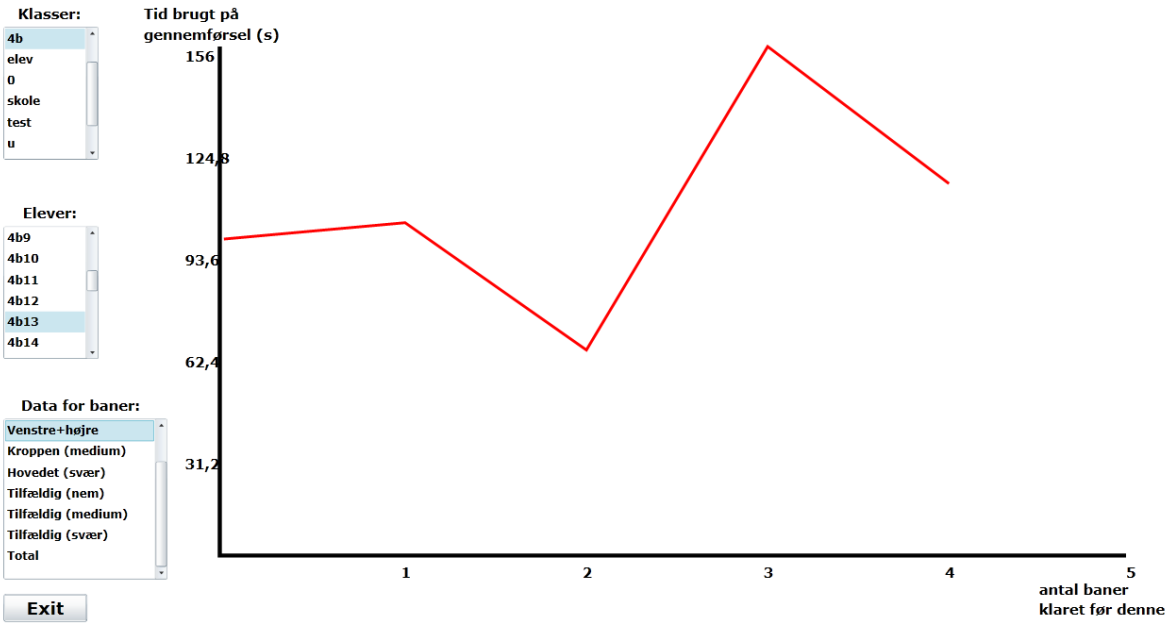


Figur 3 Brugerinterfacet til at vælge baner

Når en sektion af baner er klaret fx armen eller kroppen, vil robotten se repareret ud de steder. Dette er til for at give spilleren følelsen af at han rent faktisk reparerer robotten. På billedet ovenfor er hovedet og højre arm repareret.

2.3.1.4 Lærer klient

Ved lærer klienten er det ikke lige så vigtigt med et pænt layout og den samme atmosfære, som præger resten af spillet. Det er fordi den primære målgruppe, altså eleverne, aldrig vil komme til at se denne skærm og en lærer ikke behøves at fanges af historien på samme måde som eleverne. Skærmen indeholde 3 listbox'e hvor læreren vælger hhv. klasse, elev og kategori, hvorefter data'en vil blive vist i grafen til højre. Databasen tillader at man kan få vist data som tid brugt på bane, hvilke komponenter eleven foretrækker, og hvilke komponenter han ikke benytter sig af i lige så høj grad, men i øjeblikket er det kun muligt at se tiden der er brugt. Der findes også en knap der leder brugeren tilbage til hovedmenuen. Lærer klienten kan ses på Figur 4.



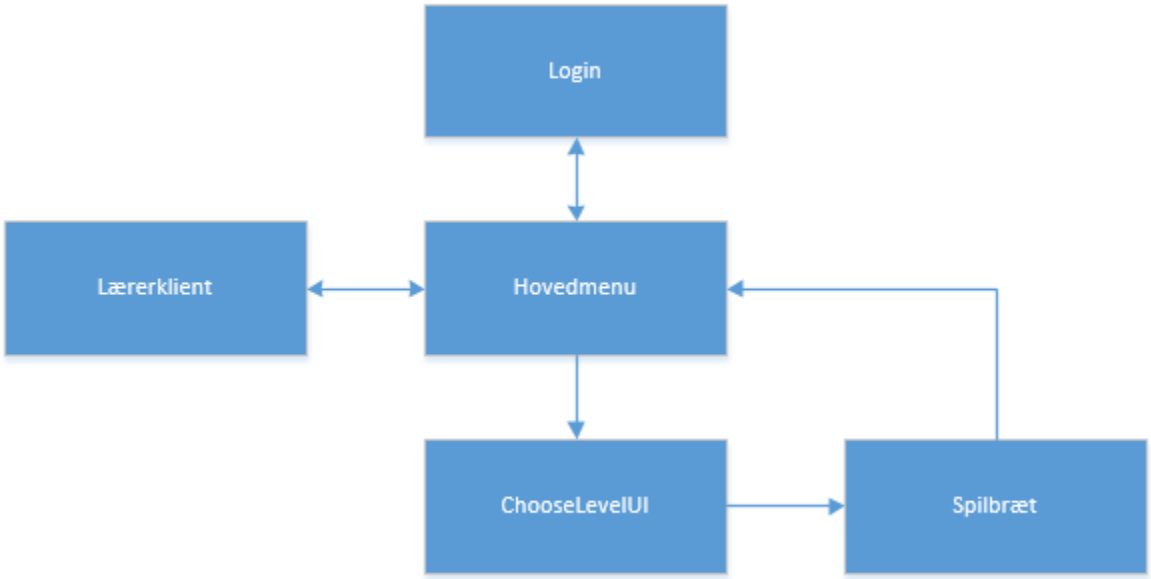
Figur 4 Et eksempel lærer klienten som viser noget data for en elev

2.3.2 Spil struktur

Spillet består af nogle forskellige menuer udover selve spilbrættet og lærerklienten. Og flowet, som er illustreret i Figur 5, mellem dem er med til at skabe et brugervenligt program.

Når spillet startes havner man i hovedmenuen. I hovedmenuen finder man en række valgmuligheder, der låses op når brugeren logger ind. Fra hovedmenuen kan man gå til Valg af Bane interfacet, som er menuen hvor man vælger baner og bliver præsenteret for den ødelagte robot. Når en bane er valgt vises spilbrættet for denne specifikke bane og brugeren har muligheden for at gå til hovedmenuen eller klare banen hvorefter spilleren bliver ledt tilbage til hovedmenuen.

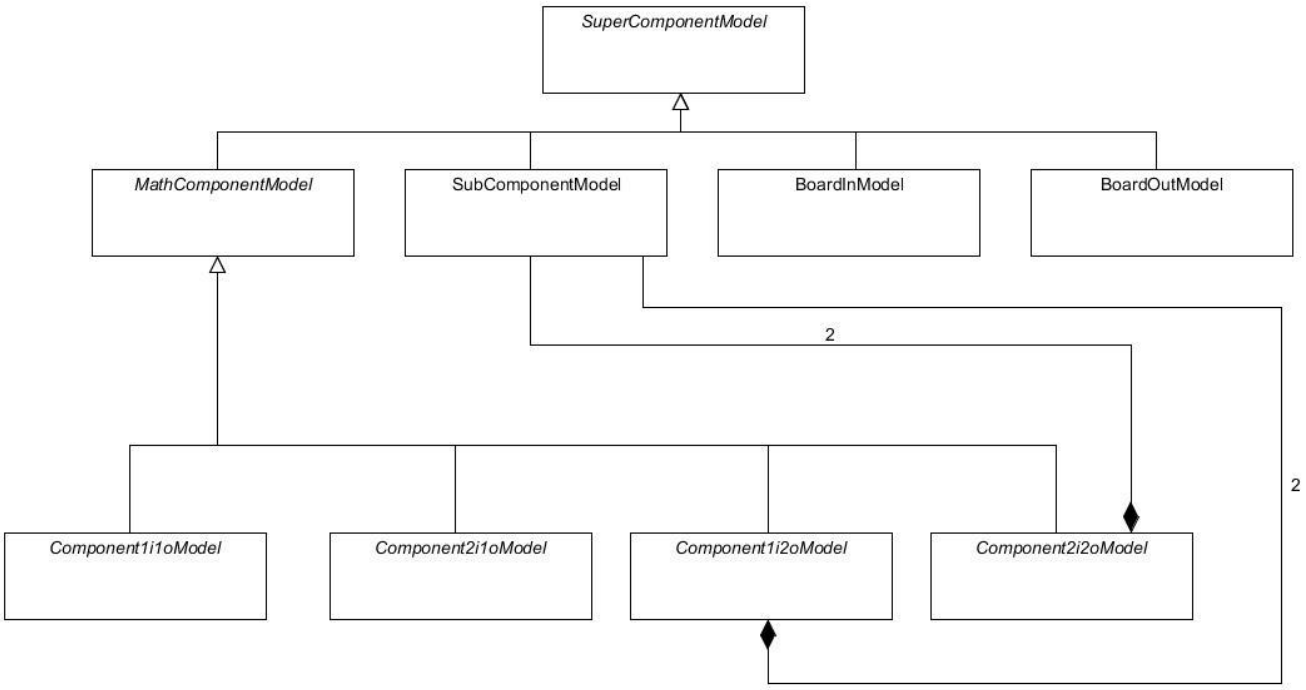
Fra hovedmenuen kan spilleren også vælge at gå til lærerklienten, såfremt brugeren er en lærer. I lærerklienten kan læreren vælge at få vist forskelligt spildata for eleverne i sin klasse. Fra lærerklienten er det muligt at gå tilbage til hovedmenuen.



Figur 5 Flow igennem spillet

2.3.3 Komponenter

De matematiske komponenter som spilleren bruger, har vi designet hierarkisk, så vi kan vedligeholde og ændre dem så simpelt som muligt, men også så vi kan håndtere dem på en generel måde og ikke skal tage hensyn til alt for mange særtilfælde.[4] Figur 6 viser den hierarkiske opbygning af komponenterne.



Figur 6 Hierarkiske opbygning af komponenterne

BoardInModel og BoardOutModel er henholdsvis komponenterne som repræsenterer de indgående tal øverste og målet nederst. SubComponentModel bliver brugt af komponenter med to udgange til at differentiere mellem de to output. Selve de matematiske komponenter som man spiller med nedarver fra en af Component1i1oModel, Component2i1oModel, Component1i2oModel eller Component2i2oModel hvor f.eks. et plus komponenter har to input og et output, så derfor nedarver den fra Component2i1oModel.

Når komponenter skal udregne deres resultat skal de bruge outputtet fra de komponenter som forbinder til den. Vi gør dette med en række rekursive kald, som for hvert komponent som er forbundet tjekker om det har hvad det skal bruge for at udregne sit resultat, hvis det har så udregner den det ved at bede om resultatet fra sine egne input komponenter. Dette system virkede godt med de første komponenter[5] vi havde lavet som alle var komponenter med et output, men da vi kom til at skulle teste det med

[4] Vi har efterfølgende fundet et smartere system som gør det endnu mere simpelt at vedligeholde, og giver os muligheden for at fjerne alle former for særbehandling. Læs mere om dette i refleksions afsnittet.
[5] Plus, minus, gange og dividere, alle komponenter med et output

komponenter som havde to output stødte vi på et problem. Vi havde nemlig ingen måde at se forskel på de to outputs og vi var derfor nødt til at indføre subkomponenter til at håndtere hvert output. Dette resulterede i, at vores system pludselig var nødt til at tage forbehold for, at for nogle komponenter skulle man have fat i deres subkomponenter for at udregne deres resultat. Dette var kun et mindre problem i starten, men som projektet skred frem blev det større og større indtil vi besluttede at vi ville prøve at designe et bedre system.

Den nye struktur ville, kort fortalt, fjerne alt under *MathComponentModel*, samt *SubComponentModel*, og i stedet designe *MathComponentModel*[6] til at kunne skifte form, udseende og matematisk funktion når som helst. Dette ville betyde at alle komponenter skulle behandles ens og vi ikke skulle tage særlig højde for komponenter med to output. Det ville også lette byrden betydeligt hvis vi ville tilføje nye komponenter, eftersom resten af vores program ikke ville se det som en ny klasse, men blot som en ny udgave af den kendte klasse. Alt i alt opfyldte dette design alt hvad vi havde håbet det oprindelig havde været, og det viste sig at det ikke ville tage lang tid at skrive. Problemet var at vi på det tidspunkt havde bygget mange af de andre systemer op efter den oprindelige komponentstruktur, og det derfor ville tage for lang tid at omskrive alle programmets andre systemer til at bruge den nye struktur, så vi endte til sidst med at beholde den oprindelige struktur og fokuserede på andre funktionaliteter i spillet.

Komponenter skifter farve afhængig af om de har de input de skal bruge for at udregne deres resultat. Dette henleder spillerens opmærksomhed på eventuelle manglende input, giver visuel feedback på brugerens handlinger når komponenter forbindes, samt hjælper på fantasien om at man sidder og arbejde på en elektrisk robot.

Komponenterne er strengt designet efter MVVM. Hver komponent har en Model til at holde interne værdier og metoder, et View som er den grafiske del som brugeren rykker rundt med, og så en ViewModel designet som Code Behind som tager sig af muse events på View'et. Alle tre dele er så sat sammen med data bindings som sørger for at alle tre dele er i samme state.

[6] Koden til dette findes i bilag 2. Den er ikke færdig, men kan give en grund ide om hvad vi havde tænkt os.

2.3.4 Komponentforbindelser

Komponentforbindelserne er relativt simpelt opbygget og deres funktionalitet ligeså. Hver komponentforbindelse er en enkel linje fra et punkt til nabo punkt. Når en ny komponentforbindelse oprettes overføres det komponent som forælderen stammer fra til barnet, så det kan give det videre til næste led, eller give det til et komponent hvis den ender i et punkt som der sidder et komponent på. For at dette simple system kan virke har vi opstillet to strenge krav til komponentforbindelser så vi sikre os det virker efter hensigten.

1. En komponentforbindelse SKAL have et kildekomponent som det stammer fra
2. En komponentforbindelse må aldrig forbinde til et komponent som selv er forbundet til kildekomponentet

Grunden til at vi har opsat 1 som et krav, er at vi kan indskrænke hvordan en komponentforbindelse skal behandles, så vi altså kan spare en masse tjeks og håndtering af specialtilfælde væk. Den beslutning har så resulteret i at hver gang et komponent rykkes skal alle komponentforbindelser som går til eller fra det komponent fjernes så vi altid sikre os en ensartet state for alle vores komponenter og komponentforbindelser.

2 skal overholdes for at vi ikke kommer ud i uendelig rekursion. Hvis komponent A beder om komponent B's output, og den har A som input, vil den så bede A om sit output for at kunne udregne sit eget output. Dette sikre vi ved at tjekke at et komponent som en komponentforbindelse er i gang med at forbinde til ikke indgår som en del af kildekomponents input.

På trods af at selve komponentforbindelsen er simpel og ikke har gennemgået mange iterationer, har måden de bruges på til gengæld. Første udkast til hvordan de skulle bruges var at man trak i pilehovedet og når man gav slip forbandt den til det nabo punkt som var nærmest der man slap. Vores begrundelse for dette valg, frem for at vælge nærmeste punkt til der hvor man slap uanset om det var nabo punkt eller ej, var at brugeren gerne ville placere sine komponentforbindelser præcist, eller efter et bestemt mønster, for at spare på pladsen. Det viste sig dog at være en enorm dårlig brugeroplevelse da vi testede det, så en bedre løsning var nødvendig.

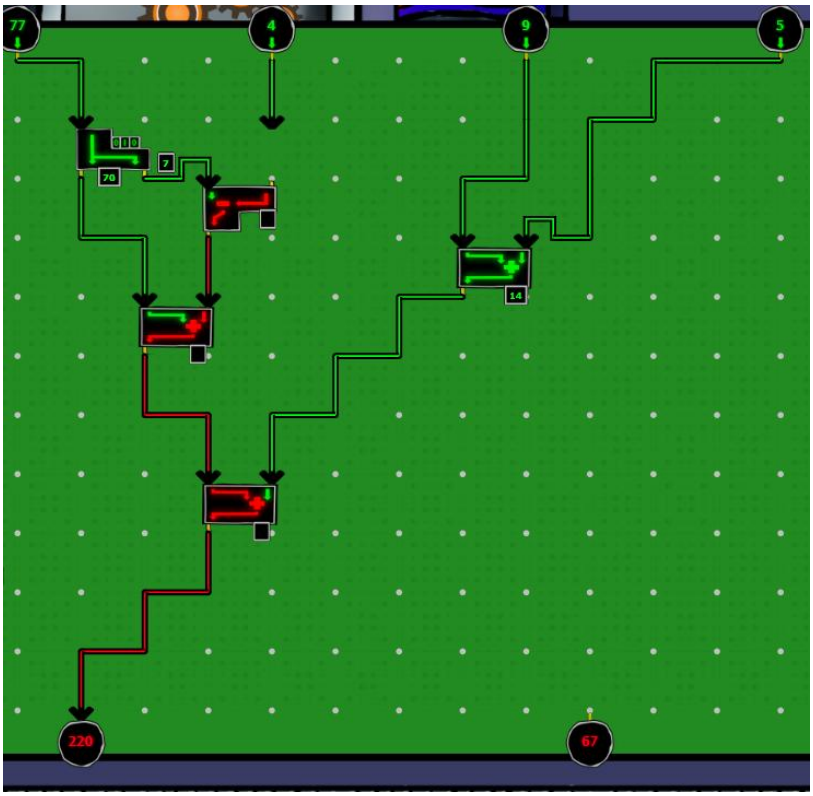
Næste iteration prøvede vi den anden ende af spektret, at uanset hvor du klikkede for at lave en forbindelse, skulle den finde vej. Første udfordring ved det var at de forskellige punkter som der skulle forbindes til mellem start punktet og det sidste punkt skulle findes af programmet selv. Vi gik derfor over til en rekursiv måde at oprette komponentforbindelser på, hvor en komponentforbindelse gik grådigt efter hvad der så ud som den bedste vej fra det nuværende punkt, og så oprettede en ny som så valgte den umiddelbart bedste vej derfra osv. Denne simple, grådige algoritme havde en masse problemer, primært at den kunne ende på et vildspor fordi den selv havde afskåret sig den rigtige vej. For at løse dette havde vi brug for en søge algoritme såsom A star (A*)[7] som kunne gennemsøge alle mulighederne men samtidig

[7] Ref 3

bruge en heuristik som ledte den i den rigtige retning for at optimere køretiden. Dette var dog for omfangende et projekt så vi indså at vi var nødt til at finde en tredje måde at gøre det på.

Den løsning vi valgte at bruge er, at hvis der eksisterer en vej med optimal afstand fra komponent-forbindelsens startpunkt til det slutpunkt man har valgt vil den kunne oprette forbindelsen, og ellers vil den fejle. Fordelen ved denne løsning er at man stadig kan nøjes med at forbinde nabo-til-nabo hvis det er hvad man ønsker. Man kan hurtigt og nemt forbinde over en strækning, så længe der intet er som blokere den, hvilket vi observerede i interne demoer der faktisk sjældent er. Og de komponentforbindelser man ikke kan lave på grund af blokeringer ville alligevel ofte give problemer, da en ikke lige vej kan være en vej som går uden om et komponent og derfor ville lukke af for fremtidige komponentforbindelser.

Vi har valgt, at komponentforbindelser skal skifte farve ligesom komponenter for at forstærke følelsen af, at man sender et elektrisk signal gennem systemet. Hvis man har en stor mængde komponenter som er forbundet og der er en enkelt forbindelse som mangler for at fuldende systemet, giver det god sporbarhed igennem alle de komponenter og komponentforbindelser man har lavet. Figur 7 illustrerer dette.



Figur 7 En enkel komponentforbindelse mangler for at fuldende et større system

Som Figur 7 også viser, opføre komponentforbindelser sig specielt hvis de forbindes til et komponent fra siden, som f.eks. det ses i højre side hvor 5 forbinder til +. For at sikre at pilehovedet for enden af en komponentforbindelse altid peger ind mod det komponent som den forbinder til har vi lavet det lille knæk på linjen, så vi altid kommer ind på et komponent foroven.

Komponentforbindelser er ligesom komponenter designet efter MVVM design pattern, og er inddelt på samme måde som beskrevet sidst i afsnit 2.3.4 Komponentforbindelser.

2.3.5 Level system

For at gøre banerne i vores spil så interessante som muligt, er det ønskeligt at hver bane har flere unikke måder at blive løst på. Vores første indskydelse var at håndlave dem, og på den måde sikre os kvaliteten. Problemet ved den fremgangsmåde er, som vi fandt ud af da vi gik i gang med at designe banerne, er at det tager meget lang tid at tjekke hver baner efter for at finde alternative løsninger. Dette kunne måske have været i orden hvis vi bare ville have haft 10 baner i alt, men 10 baner bliver hurtigt gennemført, og ved at skulle håndlave fremtidige baner blev spillet meget tungt at udvide.

Det oplagte næste skridt var at lade en algoritme søge igennem alle mulige måder at placere X komponenter på, og så til sidst lede forslagene igennem for at finde de baner med flest mulige måder at løse dem på og vælge dem. Vores oprindelige beregning for tidskompleksiteten for algoritmen var det maksimale antal komponenter vi ville bruge opløftet antallet af komponenter til rådighed i en given bane, så altså en køretid på $O(n^m)$ hvor n er antallet af komponenter vi vil bruge og m er antallet af forskellige komponenter til rådighed i en given bane. Det er ikke en god køretid men da vi kunne garantere at både n og m altid var relativt små (mellem 4-8) var det i orden. Problemet vi fandt, da algoritmen var færdig var, at vi havde glemt noget i vores tidsberegning. Vi havde antaget at et komponent kun kunne placeres et sted men i virkeligheden kan det placeres mange forskellige steder. Antaget blot fire BoardIns, kan et komponent som har to input placeres seks steder, men hvis et komponent som har et input men to output placeres først kan er der 10 mulige placeringer osv. Det betød at vi nu stod overfor en tidskompleksitet på $O((n^m)^K)$ hvor k potentielt kunne blive meget stor, og den faktiske tid det tog at generere en stor bane var nu langt over 15 minutter. Vi kan derfor ikke bruge denne algoritme ved run-time men den kunne potentielt køres uden for spillet og de valgte baner kunne manuelt indtastes. Dette ville dog også være en meget langsommelig proces, så vi valgte at lede efter en ny måde.

Vores næste tanke var, at hvis man i stedet for at lede alle mulige komponenter og placeringer igennem, bare valgte nogle tilfældige komponenter og satte dem nogle tilfældige steder for at finde en bane, ville køretiden ikke være et problem. Ideen var god og vi fik ofte interessante baner som kunne genereres på under et sekund. Grunden til vi ikke valgte denne måde at lave alle banerne på var at vi ikke kunne garantere kvaliteten af banerne eller sværhedsgraden. Det ville ikke nytte noget hvis man kunne gennemføre spillet på få minutter ved at være heldig med de baner man fik.

Til sidst gik vi tilbage til at lave banerne i hånden, men uden at tjekke lige så grundigt om hver bane har mange løsninger. De tilfældigt genereret baner bruger vi som en slags bonus baner som ikke er en del af det at reparere robotten, men med lidt fintuning kunne de bruges som "fyld" baner ind imellem de håndlavede baner i hoveddelen af spillet. Denne fintuning har vi dog valgt ikke at lave, da andre ting havde en højere prioritet.

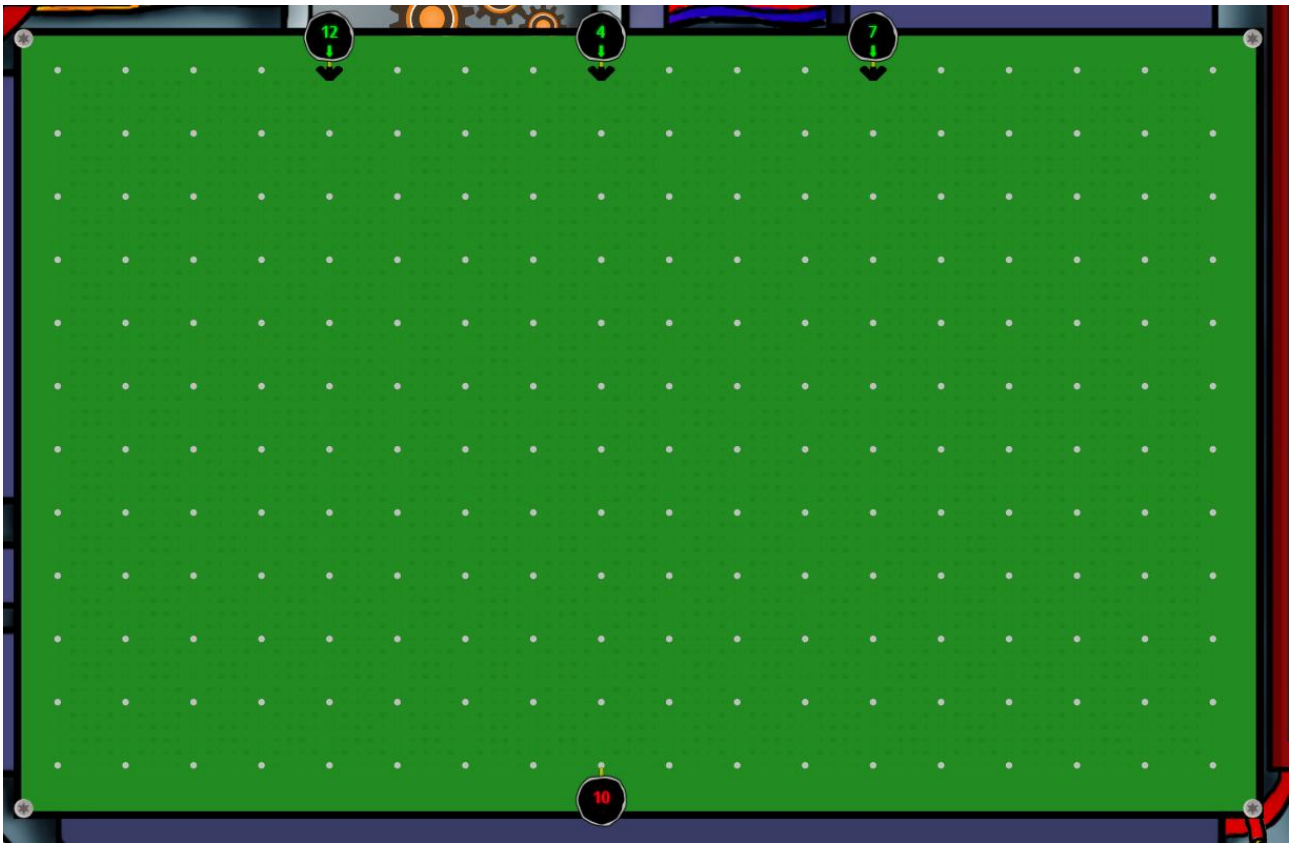
En fordel ved håndlavede baner, som vi næppe kunne opnå ved automatisk genererede baner, er også at vi her kan lave baner som tvinger eleverne ud i at skulle bruge en bestemt type matematisk viden for at

komme igennem. Og da spillet jo er et spil til at lære matematik, er det nok en god ide, som minimum, altid at beholde nogle baner som er håndlavede.

2.3.6 Spil brættet

Spilbrættet er repræsenteret af MainPage klassen og består af et array af Gridpoints og Anchorpoints, hvor Gridpoints primært har at gøre med komponentforbindelser og Anchorpoints er de punkter som komponenterne kan sidde på. Et billede af spilbrættet og den visuelle repræsentation af grid points kan ses på Figur 8.

Spilbrættet bliver genereret når et level er valgt, ud fra et LevelSpecification-objekt, der fungerer som en simpel datastruktur til at indeholde informationer om banen, såsom fikserede komponenter, komponenter i "værktøjskassen", inputs, outputs og størrelsen af spilbrættet - selvom den feature ikke bliver brugt alligevel.



Figur 8 Spilbrættet, hvor de grå prikker er der hvor der er grid points

Størrelsen af spilbrættet refererer til antallet af Gridpoints på spilbrættet vertikalt og horisontalt. Som nævnt er vi gået over til at have en fastsat størrelse spilbræt.

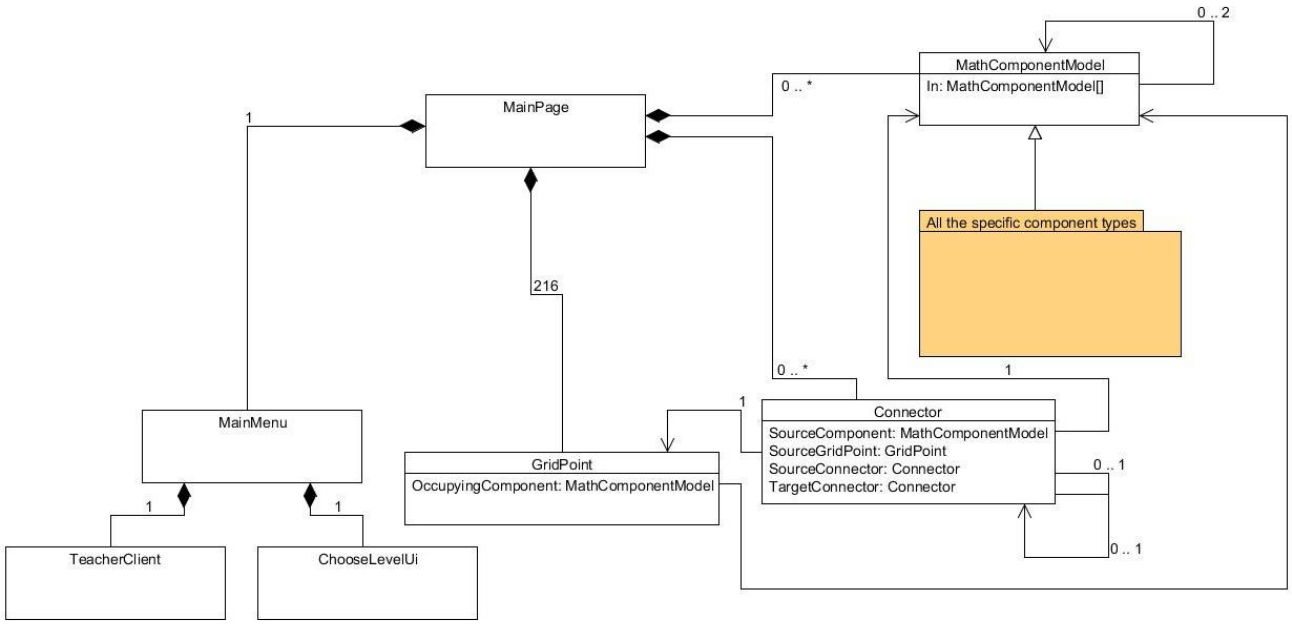
Sammen med AnchorPoints, der som nævnt beskæftiger sig med komponenterne bliver koblingen lavet gennem et såkaldt GridPointArea, som indeholder en liste med de GridPoints, der ligger rundt om

AnchorPointet. Dette gøres for at komponenterne kan modtage det input som komponentforbindelserne overfører og komponenterne kan lede deres outputværdier igennem, men bruges også til at undersøge om en plads er optaget når et komponent flyttes.

Ud fra de oprettede gridpoints bliver der oprettet GridPointArea's og AnchorPoint's for hvert af de steder et komponent kan sidde på spilbrættet. Da vores spil både har komponenter der bruger 2 og 4 GridPoint's laves der både en liste af store og små GridPointAreas og dertilhørende AnchorPoint's.

2.3.7 Component diagram

Spillets overordnede struktur er illustreret i et simplificeret component diagram på Figur 9.



Figur 9 simplificeret skematisk diagram af arkitekturen

Som det ses er MainPage det centrale element. Den faciliterer kommunikationen mellem de forskellige objekter og objekttyper, og fungerer som spillets kerne, og tillader os at holde de forskellige objekter adskilt, så de ikke kender mere til hinanden end nødvendigt.

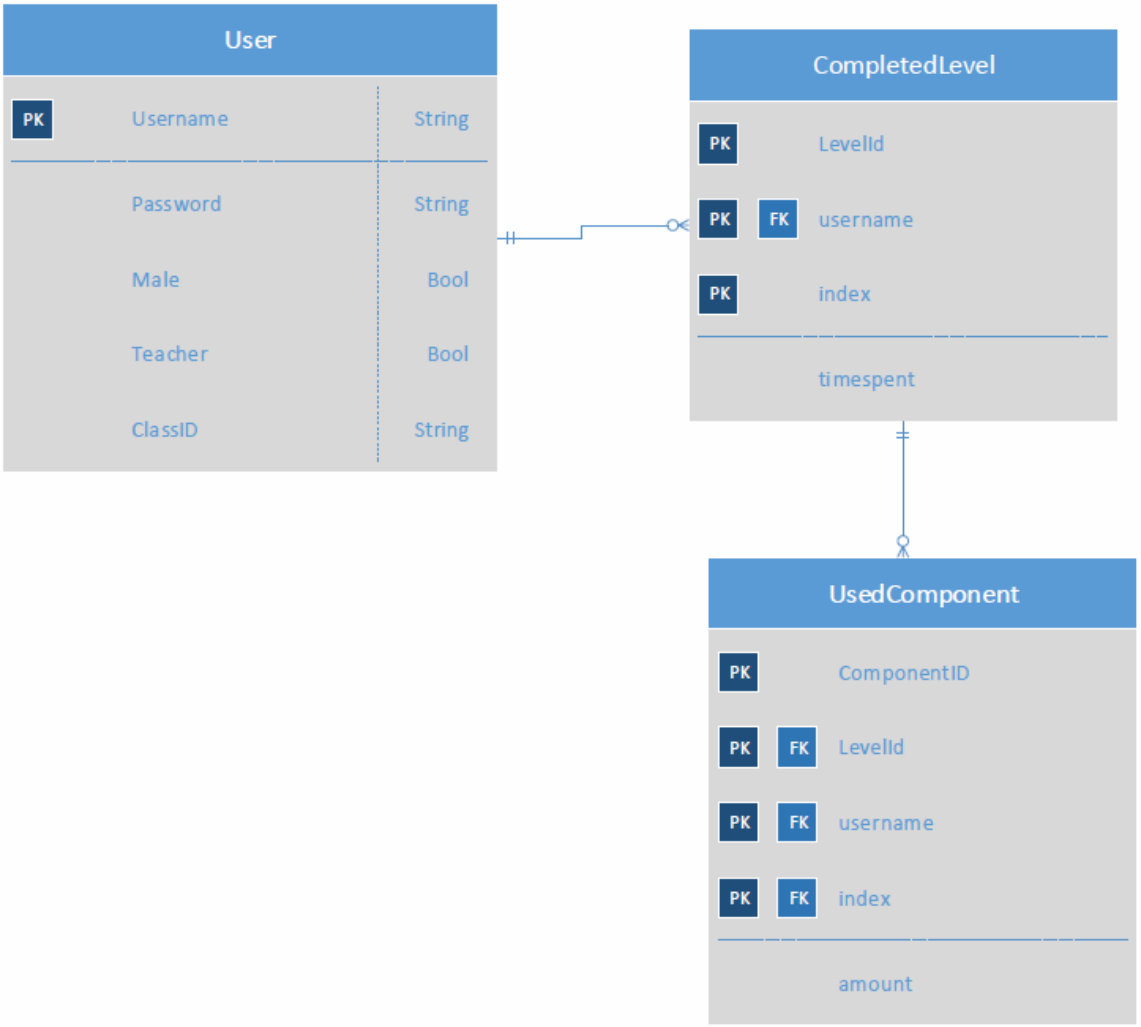
2.3.8 Database kommunikation

Databasekommunikationen bruges til at gemme data og hente den ind i programmet. Dette er nødvendigt for at tillade læreren at evaluere elevens præstation samt tillade at eleven afslutter spillet uden at skulle starte forfra fra bunden næste gang spillet startes op.

Da Silverlight kører på klienten, frem for på serveren, sætter det nogle begrænsninger for, hvordan databasen må kontaktes af programmet.

Databasekommunikation foregår ved hjælp af en GET-request til et php script, der formulerer en SQL forespørgsel baseret på parametrene fra URL'en. Der laves en separat php-fil for hver databaseoperation.

Fra Silverlight sker databaseoperationerne asynkront, hvilket betyder at databaseoperationer, hvor programmet skal vente på en slags feedback, fx login der skal tjekke om brugeroplysningerne er korrekte og meddele til programmet, hvorvidt det er tilfældet, skal standse resten af programmet. Langt de fleste af vores databaseoperationer skal ikke returnere noget, hvilket betyder at de kan køre som en separat tråd i baggrunden, mens spillet fortsætter. Databasen er meget simpel da det er meget få informationer for hver bruger, som vi har brug for at gemme. På Figur 10 ses entity-relationship modelen af databasen.



Figur 10 Entity-relationship model

Som nævnt før er en af databasens vigtigste funktioner at indeholde den data som læreren skal bruge til at evaluere en elevs spilforløb. Som indikator til læreren, hvor godt det går for eleven, bruges den tid eleven har brugt på at løse en bane.

I databasen findes entity'en User, der indeholder data om brugeren såsom brugernavn, kodeord, køn, om brugeren er lærer og et id som klassen har fået tildelt. Brugernavn er gjort til primary key, hvilket gør at to brugere til programmet ikke kan hedde det samme. Dette kunne muliggøres ved også at lade klassens id være en primary key, hvilket tillader brugere i forskellige klasser at have samme brugernavn. Vi ser det ikke som nødvendigt, da der ikke er den store risiko for at alle mulige brugernavne ville blive optaget.

CompletedLevel er en entity, der indeholder data om, hvilken bruger der har klaret banen, banens id, tid brugt på banen og et index for den klarede bane. Dette er nødvendigt for at visualisere dataen i en graf der illustrerer tid brugt for hver bane i kronologisk rækkefølge. Brugernavn tages med som foreign key, for at undgå redundans i databasen, hvilket kan føre en masse fejlagtig data, hvis der ikke tages forhåndsregler mod dem.

UsedComponent oprettes for hver type af komponent der er brugt i den klarede bane og ud over de foreign keys der er importeret fra CompletedLevel, er der også et komponentid og et antal. En anden måde at gøre det på som vi fravalgte var, i stedet for at have en række for hver komponenttype, at have en række for hver enkelt komponent. Men ved tilføjelse af et antal til entity'en reducerer vi mængden af data i databasen en hel del.

Som det fremgår af databasen er der ikke nogen personfølsomme data i spillets databasesystem. Derfor brugte vi en GET-request frem for en POST-request til at kontakte php-koden. Forskellen mellem en GET og en POST-request er at GET-requesten er langt mere usikker, da dataen der sendes ligger i URL'en. Det er altså nemt for hackere at gå ind og hive data ud af vores database. For at gøre det samme med en POST-request kræver det en smule mere kendskab til scripting og kodning, samt at det tager væsentlig længere tid at implementere, hvilket var grunden til at fravælge det, så vi stedet kunne fokusere på selve spillet. GET-requesten gør til gengæld at vi har mulighed for at afprøve vores scripts direkte i browseren.

Ved at modificere databasefunktionerne og de dertilhørende scripts en smule kan man dog sikre sig et langt højere niveau af sikkerhed. Et eksempel kunne være at lave et tjek, der sikrer at requesten kommer fra programmet frem for en browser. Det kunne være ved at kryptere programmets output eller svaret fra PHP-scriptet. Man kunne også kombinere det med en cookie der indeholdt nogen oplysninger der blev tjekket under transaktionerne.

3 Implementering

I dette afsnit vil vi gennemgå nogle af de interessante implementeringer vi har lavet. Vi ville ikke lave en line-by-line gennemgang af koden, men i stedet forklare om det på et højere abstraktionsniveau selvom det stadig vil være mere kode nært end i afsnit 2.3 Design.

3.1 Database og web integration

Til implementering af DatabaseFunctions metoderne bruger vi en udvidelse af Silverlight API'et kaldet "Async for .NET Framework 4, Silverlight 4 and 5, and Windows Phone 7.5 and 8"[8] til at gøre arbejdet med asynkrone funktioner mere intuitiv. Pakken tillader at man bruger keywordet await til at vente på et asynkront Task-objekt er klar med et resultat og derefter returnere Task-objektets resultat.

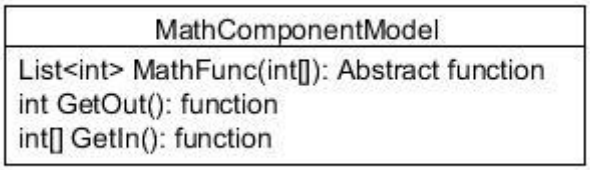
Klassen DatabaseFunctions bruges som wrapperklasse til en række statiske funktioner, der har til formål at sende HTTP get-requests til php filerne på serveren.

PHP scriptet opretter en forbindelse til en mysql database ved at bruge funktionen mysqli_connect[9], der tager imod adressen på den server databasen ligger på, et brugernavn, et kodeord og navnet på databasen, som parametre og returnerer en MySQL link identifier.

For at sende en forespørgsel til databasen bruges funktionen mysqli_query[10], der tager imod en MySQL link identifier og en string med SQL forespørgslen. Hvis forespørgslen var en select returneres resultatet række for række ved hjælp af funktionen mysqli_fetch_array der tager et objekt af typen mysqli_result som parameter og returnerer et string array.

3.2 Komponenter

Det som primært er interessant ved vores implementation af komponenter, udover den hierarkiske opbygning som blev diskuteret på Figur 6 i afsnit 2.3.3, er måden vi håndtere input og output på. Et komponent har et sæt af generelle metoder til at udregne sit resultat. Et meget forsimplet klasse diagram som viser disse ses i Figur 11.



Figur 11 Et meget forsimplet klassediagram til demonstration af input/output

MathFunc er en abstrakt metode som hvert specifikke komponenter definere og som afgør hvilken matematisk funktion som komponenten udføre. *GetOut* kalder *GetIn* for at få komponents input og

[8] Ref 4
[9] Ref 5
[10] Ref 6

returnere så komponentens eget output. *GetIn* sætter en rekursion i gang som kalder *GetOut* for de komponenter som forbinder direkte til den selv, og på den måde løber vi alle komponenter som sidder sammen igennem og udregner deres output. Vi kan på den måde automatisk opdatere hele systemet hver gang en ændring foretages, hvilket styres fra metoden *EvaluateSystem()* som findes i MainPage. Vi har ligeledes taget en rekursiv tilgang til at tjekke om et komponent har alle sine inputs opfyldt så den kan lave sit output, hvilket tjekkes i *EvaluateSystem* inden *GetOut* kaldes. Fremgangsmåden er den samme som ved *GetOut*, hvor den først tjekker om den selv har alle sine inputs, hvorefter den tjekker om sine inputs har inputs og så videre.

3.3 Level generation

I dette afsnit vil vi gennemgå først strukturen af levels, og dernæst kigge lidt nærmere på algoritmen som laver dem, både den originale og den vi endte med.

3.3.1 Strukturen af levels

Til implementationen af spillets baner har vi en speciel klasse, *LevelSpecification*, som definere hvordan en bane ser ud. Den indeholder ikke blot hvad der er inputs og outputs for banen og hvilke komponenter der er til rådighed, men også alle de interne informationer om en bane, såsom om dens level id, hvilken sektion den høre til, om den er autogeneret og level id'er på de baner man skal have klaret for at kunne spille den. Derudover har *LevelSpecification* også to vigtige *public static* metoder:

- *GenerateRandomLevelSpec(int difficulty, int lvlIdToUnlock)*
- *GetLevelSpecsForSection(enum WhichSection)*

Den første kalder *SolutionWizard* som laver en tilfældig, dynamisk genereret bane ud for den givne sværhedsgrad, sætter så den level id som er påkrævet for at kunne spille den og returnere så en *LevelSpecification*.

Den anden laver alle banerne for en given sektionen. De er i forvejen indtastet i en anden klasse, *LevelCollection*, hvilket gør det mere overskueligt at håndtere og lave ny baner. LevelCollection repræsentere baner i deres pureste form, hvor kun input, output, komponenter og fikserede komponenter er indtastet. Der er en liste for hver sektion som indeholder disse og på den måde er der indirekte viden om banens sektionen og placering i sektionen. Banerne i *LevelCollection* er blot en funktion som returnere de førnævnte ting, og som så bliver brugt af *GetLevelSpecsForSection* til at oprette et realt *LevelSepcification* som spillet så kan indlæse.

Spillet bruger kunne disse to metoder til at oprette baner, og aldrig direkte constructor'en for *LevelSpecification*.

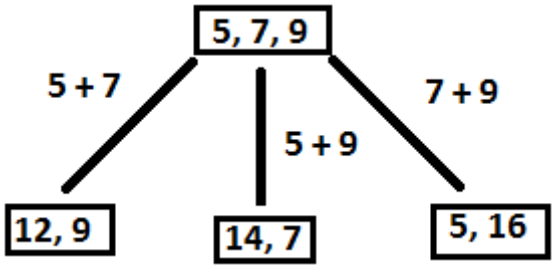
Den oprindelige ide ved at have *LevelSpecification* strukturen var at spilleren eller læreren selv kunne oprette baner, enden for sjov eller for at træne en elev på et specifikt område. Ideen var at spillet havde en editor man kunne åbne og indtaste sin input og output i, samt hvilke komponenter der var til rådighed og så gemme det ned i en fil, enden på serveren eller på brugerens computer. Spillet ville så ved hjælp af

serialization indlæse filerne og ligge det ind i et *LevelSpecification* objekt, som vi så kunne bruge til at oprette banen inden i spillet med vores standard procedure. Denne feature ende dog med at blive nedprioriteret til fordel for andre features, men grundarbejdet og strukturen er lavet og gør det nemt at potentielt implementere i fremtiden.

3.3.2 Automatisk genereret levels

Som forklaret i afsnit 2.3.5 Level system var den første iteration af level genererings algoritmen for grundig og tog derfor for lang tid at køre, og vi vil derfor ikke komme nærmere ind på den. Dog er den grundlæggende ide den samme for både den originale algoritme og den vi endte med at bruge, så derfor bliver ideen kort opridset her.

Ideen kom fra et møde med vores vejleder Jeppe Revall Frisvad hvor vi snakkede om måder at automatisk genererer baner, og der kom han med den ide som senere udviklede sig til den algoritme vi bruger nu. Algoritmen kunne vælge nogle tilfældige tal som inputs og fik et sæt af komponenter som vi havde bestemt afhængig af sektion og bane nummer. Den ville så forgrene sig ud som det ses på Figur 12 for hver komponent til rådighed på enhver placeringer og algoritmen ville så rekursivt prøve alle komponenter igen på de nye tal som var til rådighed.



Figur 12 Med tallene 5, 7 og 9 til rådighed bliver et plus komponenter placeret alle mulige steder

Efter hver rekursion ville de tal som var til rådighed blive lagt ind i en ny liste som blev gemt og når rekursionen havde nået en bestemt dybde, som var bestemt afhængig af bane nummer og sektion, ville den enkelte gren terminere en anden forgrening ville blive kørt.

Alle listerne ville så blive sammenlignet og på den måde ville man kunne finde det sæt af tal som der var flest måder at opnå på og dem valgte man så som output. Derved kunne vi opfylde vores ønske om mange forskellige måder at klare en bane på. Men som beskrevet tidligere tog denne algoritme for lang tid og vi valgte i stedet at lave en ny version som er den *SolutionWizard* bruger til at oprette tilfældigt genereret baner i det endelig spil.

Fremgangsmåden er næsten den samme. Vi starter med nogle tilfældige inputs, men i stedet for at have et forudbestemt sæt af komponenter kan alle vælges. Selve algoritmen vælger et tilfældigt komponent og forbinder det med et eller flere tilfældige tal, fjerner de brugte tal fra de tal som er til rådighed, og tilføjer så komponentets resultatet / resultaterne. Så i stedet for at tjekke alle tænkelig veje ned igennem træet, vælges en enkelt tilfældig vej blot, og den bliver så returneret uden videre inspektion. Der er dog nogen

checks undervejs som sikre en vis kvalitet. F.eks kan et filter100 komponent ikke vælges hvis ikke der er nogen tal på 101 eller mere, og det kan kun blive brugt på de tal.

3.4 Spilbrættet

Generering af spilbrættet foregår i CreateLevelFromSpecs metoden i code-behind til MainPage, gennem en række trin.

Metoden kaldes først og fremmest med en LevelSpecification som parameter. Herfra fås størrelsen af spilbrættet, som gemmes i to variable, dimX og dimY, lokalt i metoden og overføres til felterne hhv. _boardSizeX og _boardSizeY, i klassen. Disse bruges som reference i andre metoder der er afhængige af spilbrættets størrelse. Dette er et levn fra tidligt i projektforløbet, hvor det var tænkt at der kunne være flere størrelser af spilbræt.

Dernæst tilføjes GridPoints ud fra bredden og højden af spilbrættet og antallet af GridPoint's som den førnævnte LevelSpecification angav således at de fordeles jævnt på spilbrættet. For hver GridPoint på nær kolonnen yderst til højre og nederste række, tilføjes et "tile"-billede. Dette billede udgør detaljerne på printpladen mellem 4 gridpoints. Højden og bredden sættes til afstanden til næste GridPoint hhv. vertikalt og horisontalt.

Efter alle GridPoints er på plads gennemløbes de igen, på nær nederste række. For hver af de gennemløbne GridPoints skal der oprettes et GridPointArea, indeholdende GridPointet selv og det Gridpoint der ligger direkte nedenunder, samt der skal oprettes et AnchorPoint beliggende midt imellem de to. De to nye objekter bliver tilføjet til listerne hhv. GridPointAreasSmall og SmallAnchorPoints, der indeholder GridPointAreas og AnchorPoints der har at gøre med de komponenter der består af en enkelt indgang og udgang. Såfremt GridPointet ikke befinder sig i sidste kolonne, tages også de to GridPoints til højre for de der udgjorde det lille GridPointArea og til sammen dannes et nyt GridPointArea og dertil også et AnchorPoint i midten af de fire. De to nye objekter tilføjes til listerne GridPointAreasLarge og AnchorPoints, der beskæftiger sig med de store komponenter.

Efter det er gjort oprettes ToolboxAnchorpoints, og de komponenter som LevelSpecification-objektet angiver skal ligge i værktøjskassen bliver oprettet i 2 versioner. En model og en dummy, hvor modellen er det objekt brugeren rent faktisk kan tage og flytte på og dummyen vil ligge tilbage så det ikke ser ud som om komponentet forsvinder fra værktøjskassen når man flytter det.

Komponenter tilføjes til spilbrættet ved at tilføje dem til en ObservableCollection, der er defineret i MainPage'ens code-behind. ObservableCollection er en datastruktur der fungere som en list men som implementerer interfacet INotifyCollectionChanged og kan bindes til et ItemsControl. Når en model bliver tilføjet til ObservableCollection notificere den vores MainPage der opretter viewet, ud fra en mapping mellem model og view, for den model på den placering i strukturen, hvor dens ItemsControl er placeret.[11]

Udover almindelig komponenter, bliver fikserede komponenter, banens input og output, samt komponentforbindelser tilføjet til spilbrættet på den måde.

[11] Ref 7

4 Resultater

Vi vil i dette afsnit gennemgå de resultater vi har fået fra vores test på Fuglevangsvej skolen. Vi foretog test af spillet i en 2. klasse, 3. klasse og en 4. klasse, men mere om dette i det pågældende underafsnit. Vi fik desuden kunden for vores projekt Erik Ottar Jensen, som er lærer på skolen, til at sende sin feedback fra at have observeret sine elever spille spillet. Til slut har vi en use case af spillet hvor alle funktionaliteter bliver afprøvet og dokumentere så det kan ses at produktet virker som forventet.

4.1 Use cases

Vi vil her demonstrere nogle generelle use cases som viser at programmet opføre som det skal ved alle handlinger. Vi har dokumenteret det med screenshots fra spillet, hvilket kan efterprøves med det brugernavn og password som forefindes i README filen som er pakket sammen med koden.

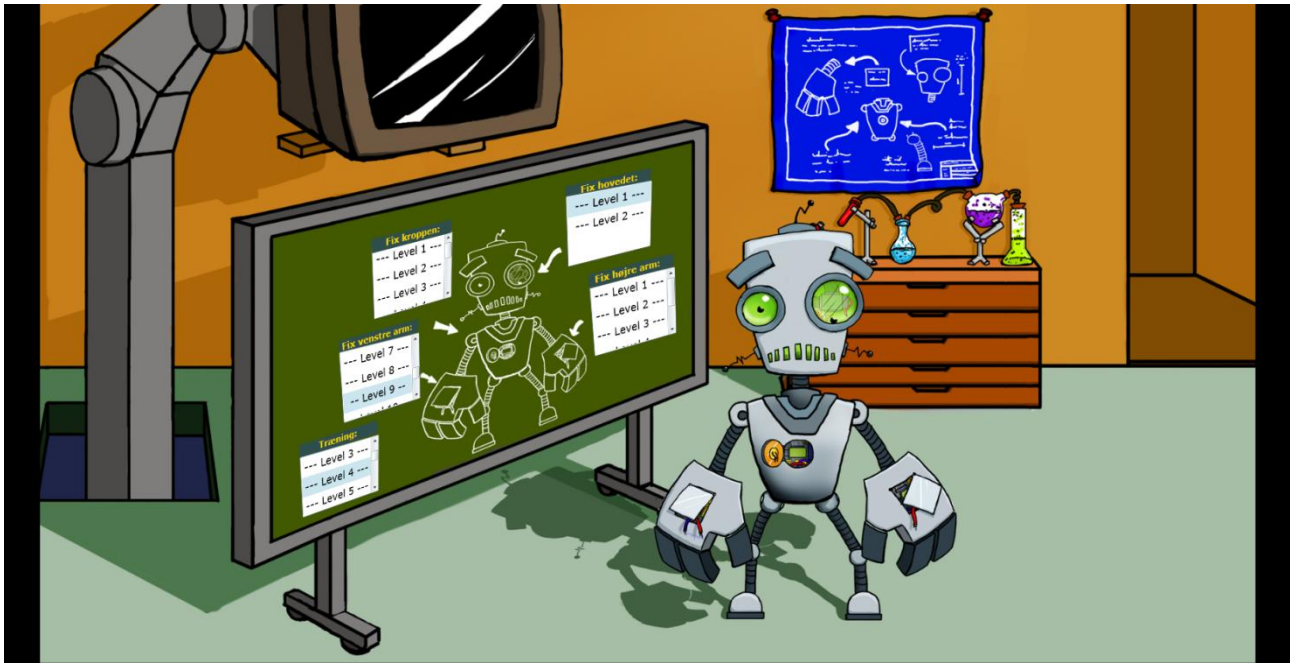
I Figur 13 ses et forsøg på at logge ind med et forkert password. Som forventet bliver man afvist.

Figur 13 Forkert password

I Figur 14 bruges det rigtige password nu. Man bliver logget ind som forventet.

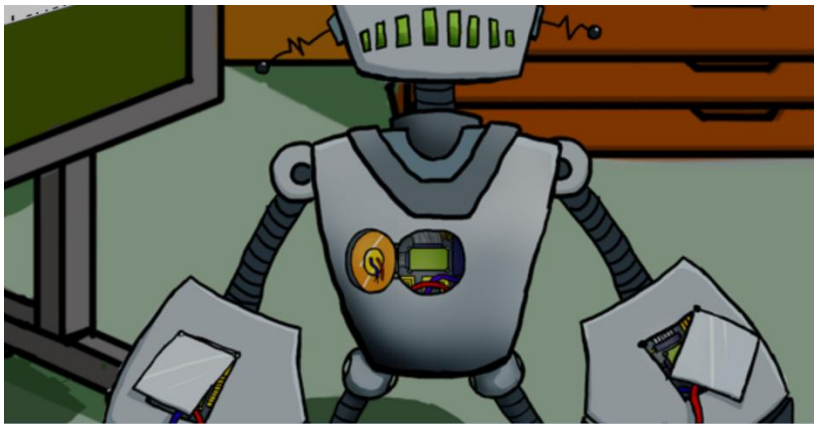
Figur 14 Rigtigt password

I Figur 15 er "Vælg en bestemt bane" knappen valgt. Man bliver som forventet præsenteret for interfacet til at vælge baner.



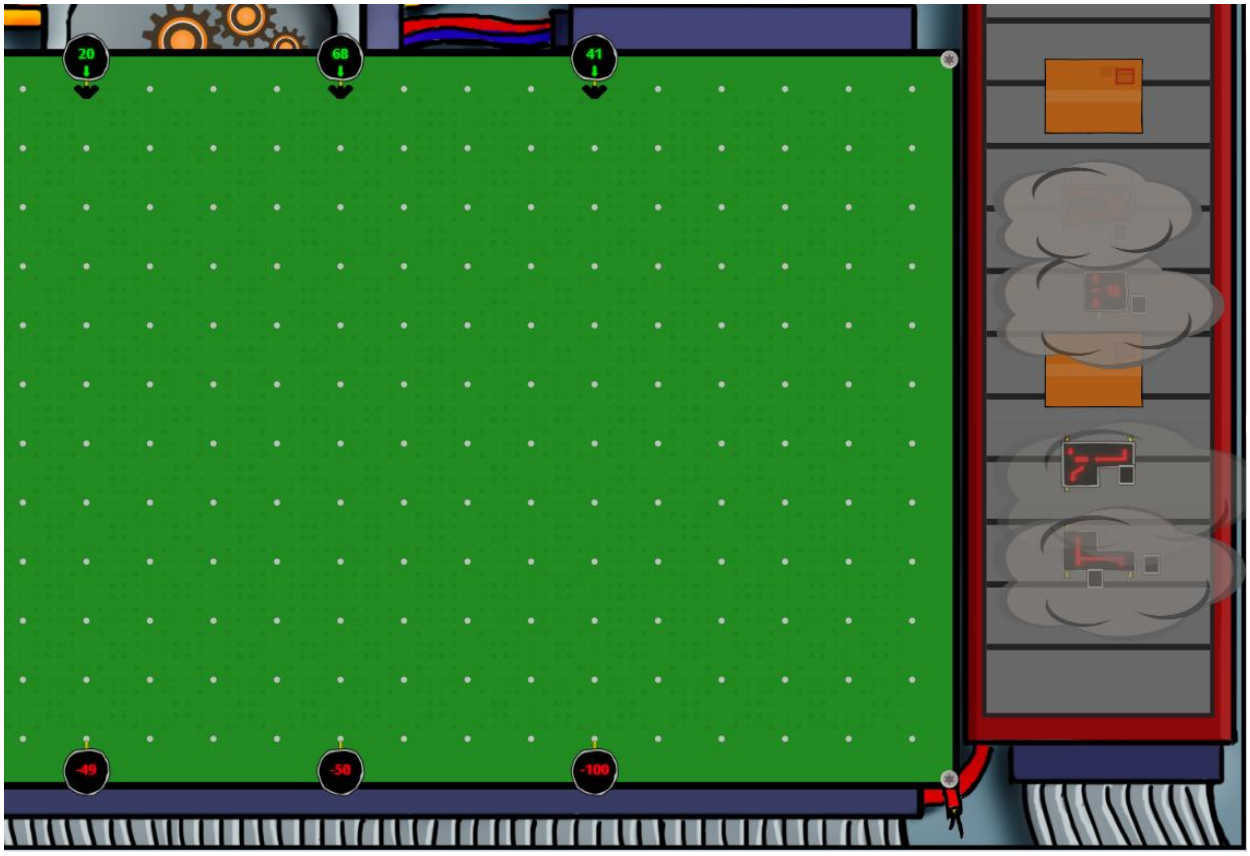
Figur 15 Valg af bane

I Figur 16 er level 1 i kroppen valgt. Kameraet zoomer ind på kroppen og fader til sort som forventet.



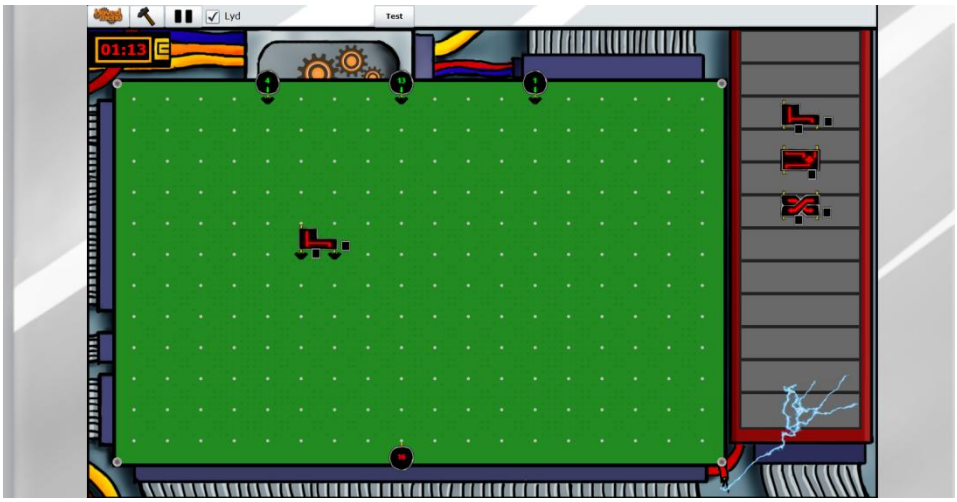
Figur 16 Zoom og fade effekt

I Figur 17 er en bane valgt (i dette specifikke tilfælde en tilfældig genereret bane af sværhedsgrad svær) og spillet har netop fadet tilbage. Grunden til animationen vises i denne bane frem for level 1 i kroppen, er at den illustrere animationen bedre og giver et mere klart billede af hvordan den ser ud. Som forventet loader banen, og intro animationen hvor komponenterne køre ind på transportbånd i kasser, som sprænger i luften, afspilles.



Figur 17 Intro animation

I Figur 18 ses level 1 i kroppen. Brugeren har taget musen henover et komponent, holdt venstre museknap nede, bevæget musen ind over den grønne spilleplade og sluppet komponentet. Som forventet lykkes det og komponent finder sig til rette så den passer i nettet af prikker.

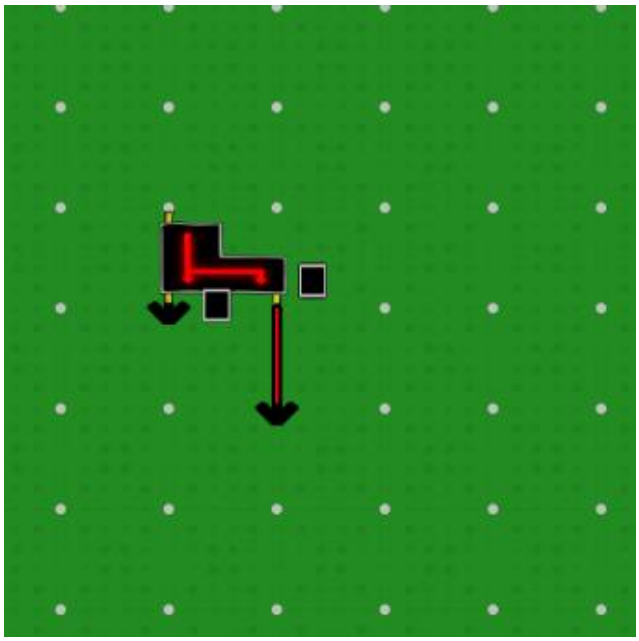


Figur 18 Drag-n-drop af komponent, samt snap to grid

Figur 19 viser hvad der sker hvis man klikker på "Gå til hovedmenu" knappen. Som forventet advares bruger om at hvis dette gøres vil man ikke kunne spille denne bane videre hvor man var kommet til.

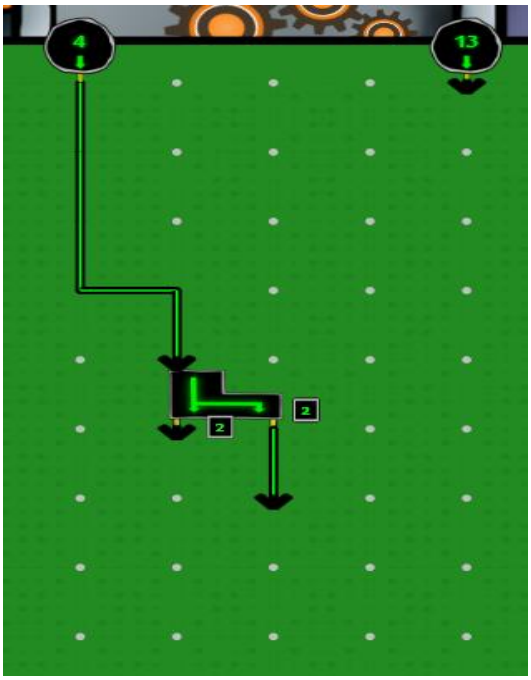
Figur 19 Klik på "Gå til hovedmenu" knappen

I Figur 20 trækkes der i en komponentforbindelse, hvilket fungere ligesom drag-n-drop. Når musen slippes sætter komponentforbindelsen sig fast på et punkt.



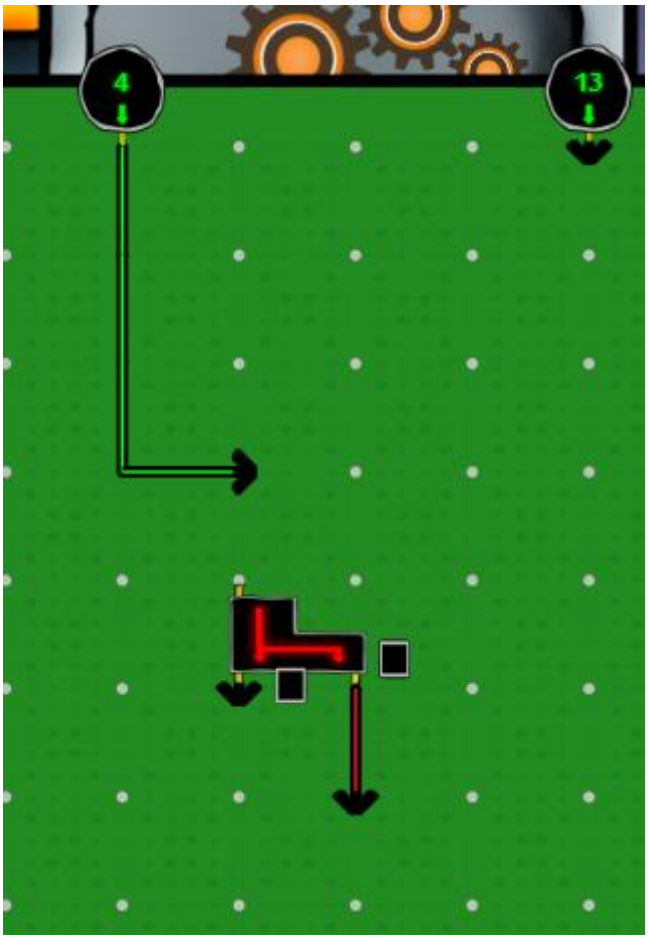
Figur 20 En komponentforbindelse bliver trukket

I Figur 21 **Fejl! Henvisningskilde ikke fundet.**bliver en komponentforbindelse forbundet til et komponent. Som forventet registrerer komponenten komponentforbindelsen, bliver grøn for at symbolisere der er signal igennem, udregner sine resultater og viser dem i sine outputs.



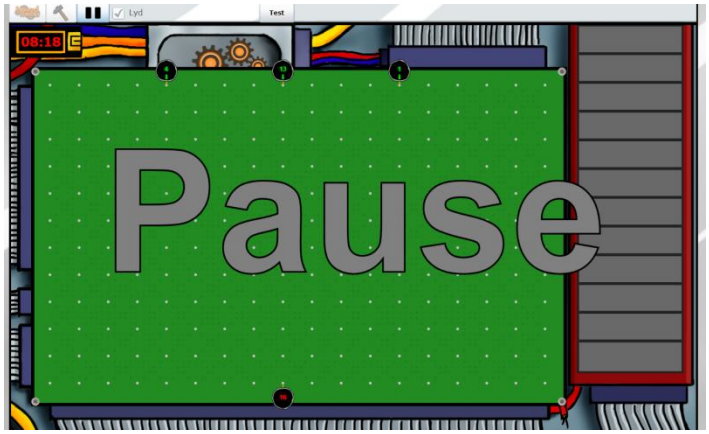
Figur 21 Komponentforbindelse bliver forbundet til et komponent

I Figur 22 er hammerværktøjet, som bruges til sletning af komponentforbindelser, øverst i skærmbilledet valgt og brugt på den nyoprettede komponentforbindelse. Som forventet slettes komponentforbindelsen, og komponenten har ikke længere det den skal bruge for at udregne sine resultater og skifter farve til råd.



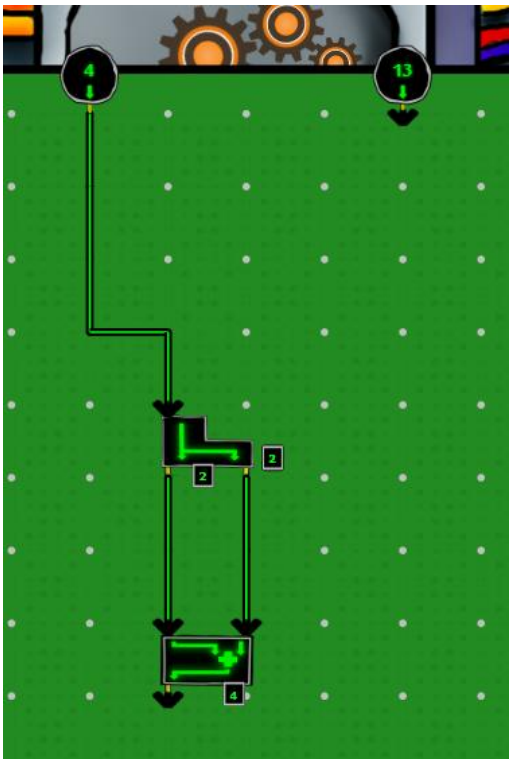
Figur 22 Sletning af komponentforbindelse

I Figur 23 er spillet blevet sat på pause oppe fra menuen øverst på skærmen. Som forventet bliver der skrevet Pause på skærmen, og alle komponentforbindelser og komponenter, inkl. dem som ikke er sat på spilpladen, forsvinder så brugeren ikke kan prøve at regne banen ud, uden at uret tikker. Ligeledes bliver de resterende knapper i menuen gjort utilgængelige (med undtagelse af Test knappen som ikke gør noget når spillet køres online).



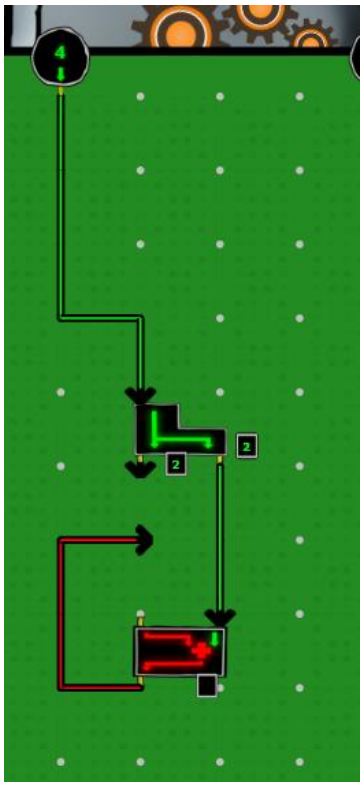
Figur 23 Spillet bliver sat på pause

I Figur 24 er spillet genoptages og et nyt komponent er blevet sat og forbundet med komponentforbindelser. Som forventet bliver den grøn når den får alle sine inputs og udskriver sit resultat, som afhænger af resultatet af det forrige komponent.



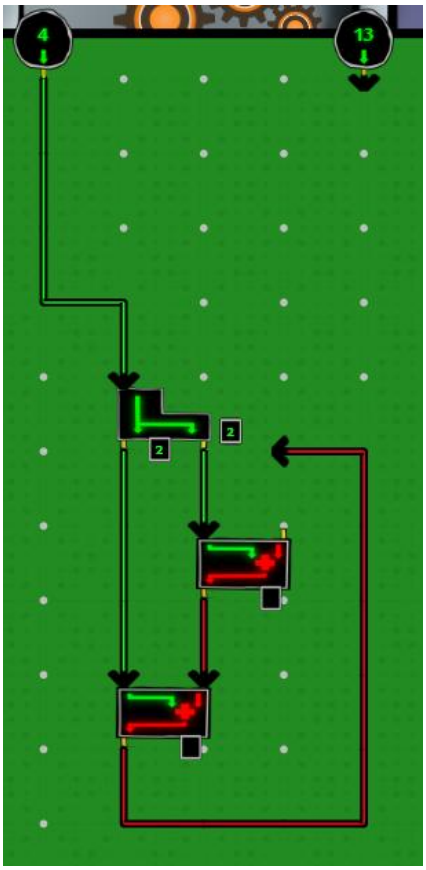
Figur 24 Komponent forbundet til et andet komponent

På Figur 25 er det forsøgt at forbinde et komponent til sig selv. Som forventet kunne dette ikke lade sig gøre og komponentforbindelsen gik tilbage til sin udgangsposition da den blev sluppet over sit eget komponent.



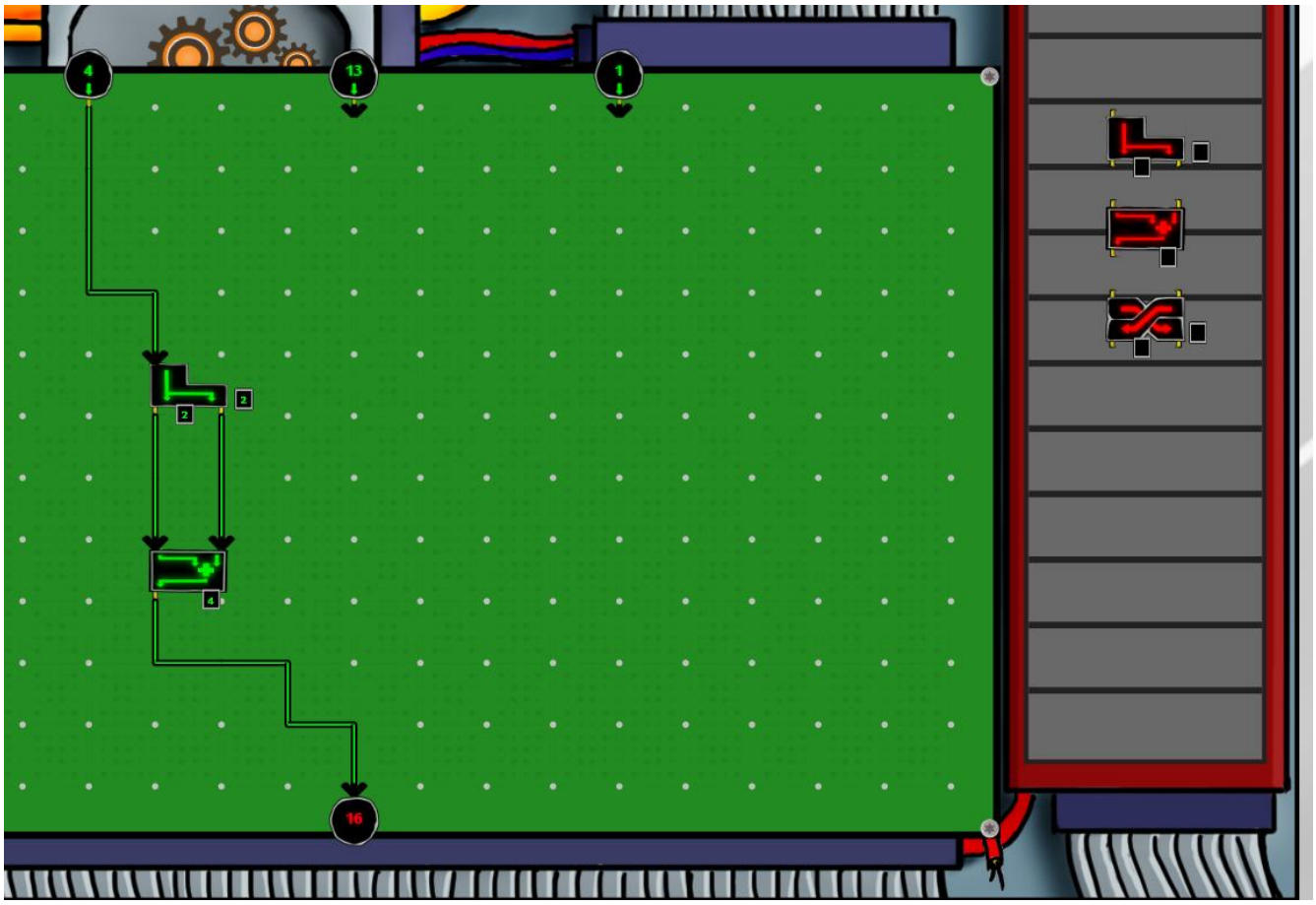
Figur 25 Komponent forsøgt forbundet til sig selv

På Figur 26 er et komponent forsøgt forbundet til et komponent som ikke er den selv men som den selv afhænger af. Som forventet kunne dette ikke lade sig gøre og komponentforbindelsen gik tilbage til sin udgangsposition da den blev sluppet over komponenten som den selv afhang af.



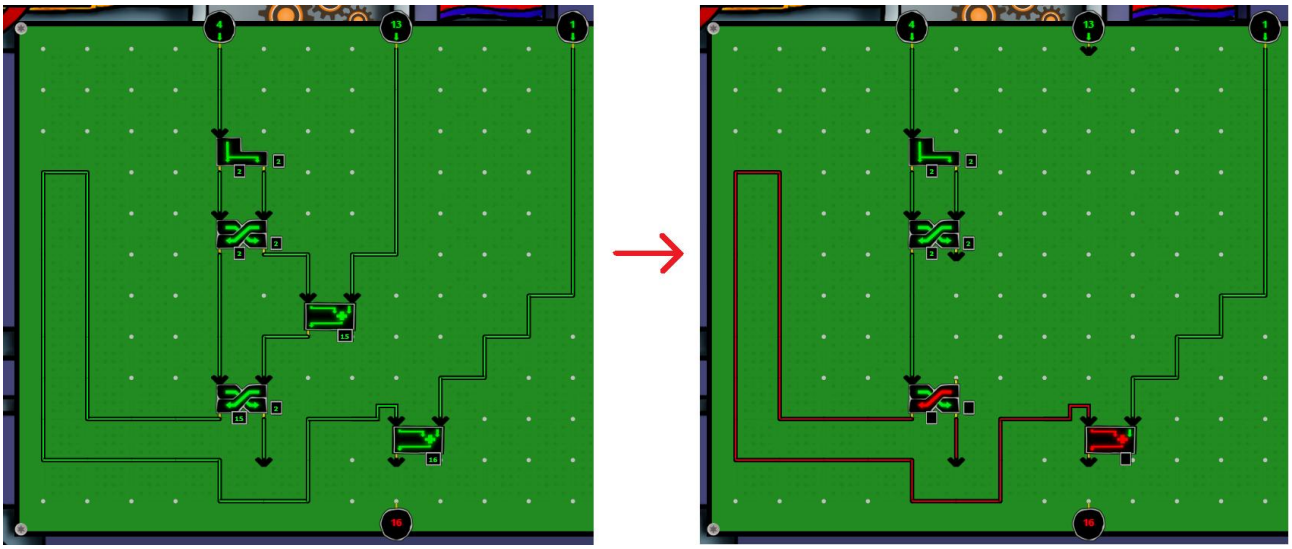
Figur 26 Komponent forsøger at forbinde til et komponent som ikke er den selv, men som den selv afhænger af

Figur 27 viser et forsøg på at gennemføre en bane ved at forbinde et forkert tal til spillets output. For som forventet sker der intet og tallet er stadig rødt.



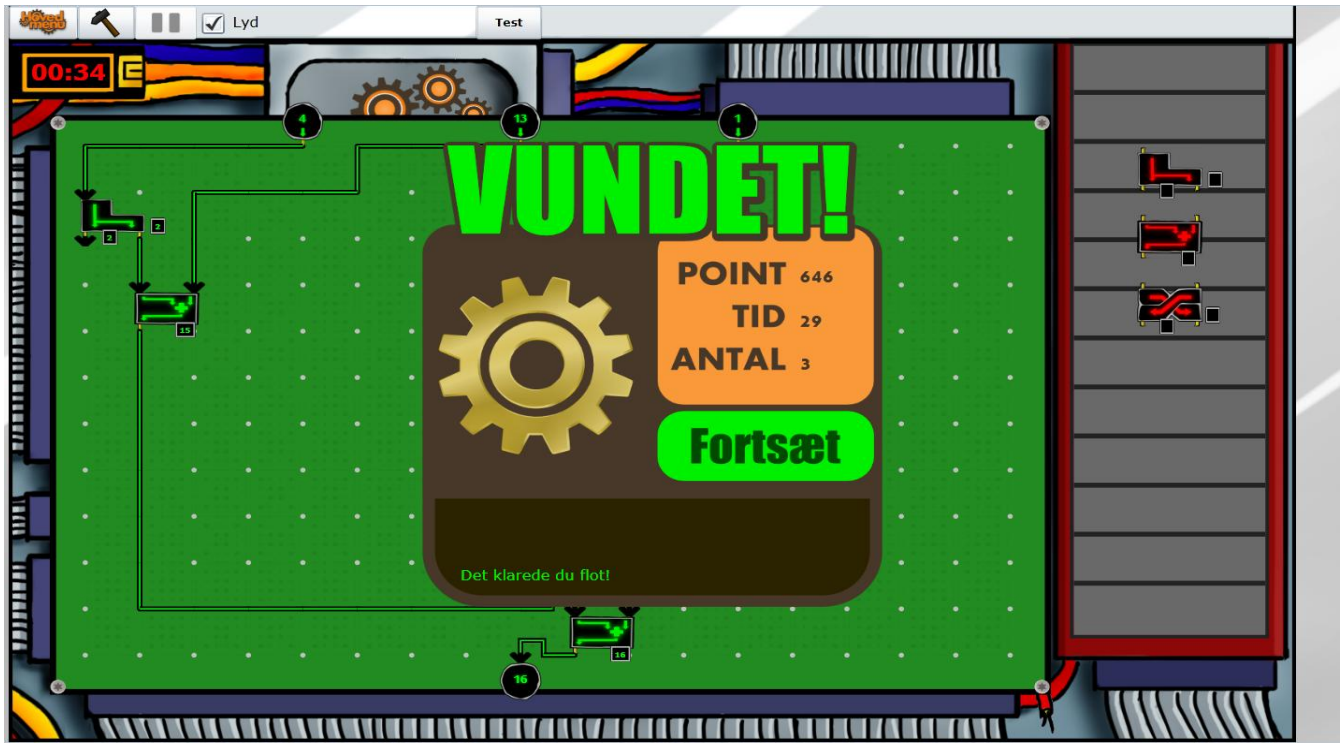
Figur 27 Forsøg på at gennemføre en bane med forkert tal

På Figur 28 er er der et før og efter billedet, der viser hvad der sker når et komponent fjernes og det er forbundet til andre komponenter og andre komponenter er forbundet til det. Som forventet fjernes alle komponentforbindelser som går til og fra komponenten, og alle komponenter som afhang af dets resultat kan nu ikke længere udregne deres eget resultatet og bliver derfor røde.



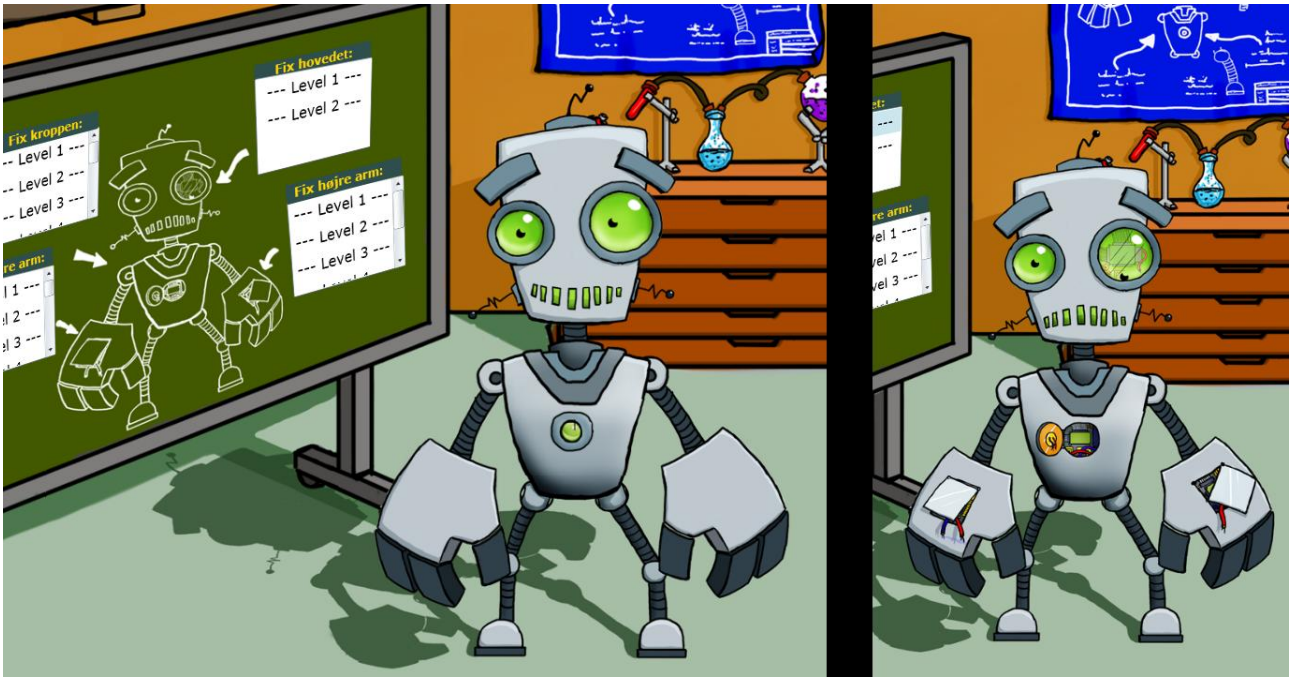
Figur 28 Fjernelse af komponent med komponentforbindelser til og fra det

Figur 29 demonstrerer, hvad der sker når alle spillets output er opfyldt. Som forventet bliver point, tid og antal komponenter vist i en scorescreen. Klikkes på fortsæt vil samme rutine som "gå til hovedmenu" knappen blive kørt, bortset fra den springer checks over som ville spørge om du er sikker.



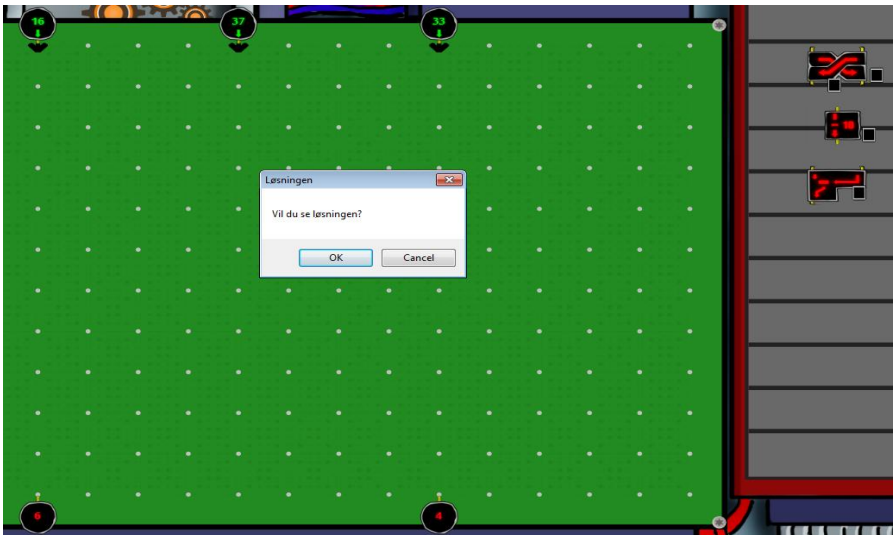
Figur 29 Et spil er vundet

Når alle baner i en given sektion er gennemført vil den kropsdel på robotten som sektionen tilhøre være visuelt repareret. Når alle sektioner er gennemført og den er fuldt repareret vil robotten også smile i stedet for at være ked af det. På grund af en mindre bug i login systemet kan denne features desværre ikke testes online, men Figur 30 viser den fulde repareret robot kørt i offline mode fra en af vores maskiner, med den helt ødelagte ved siden af til sammenligning.

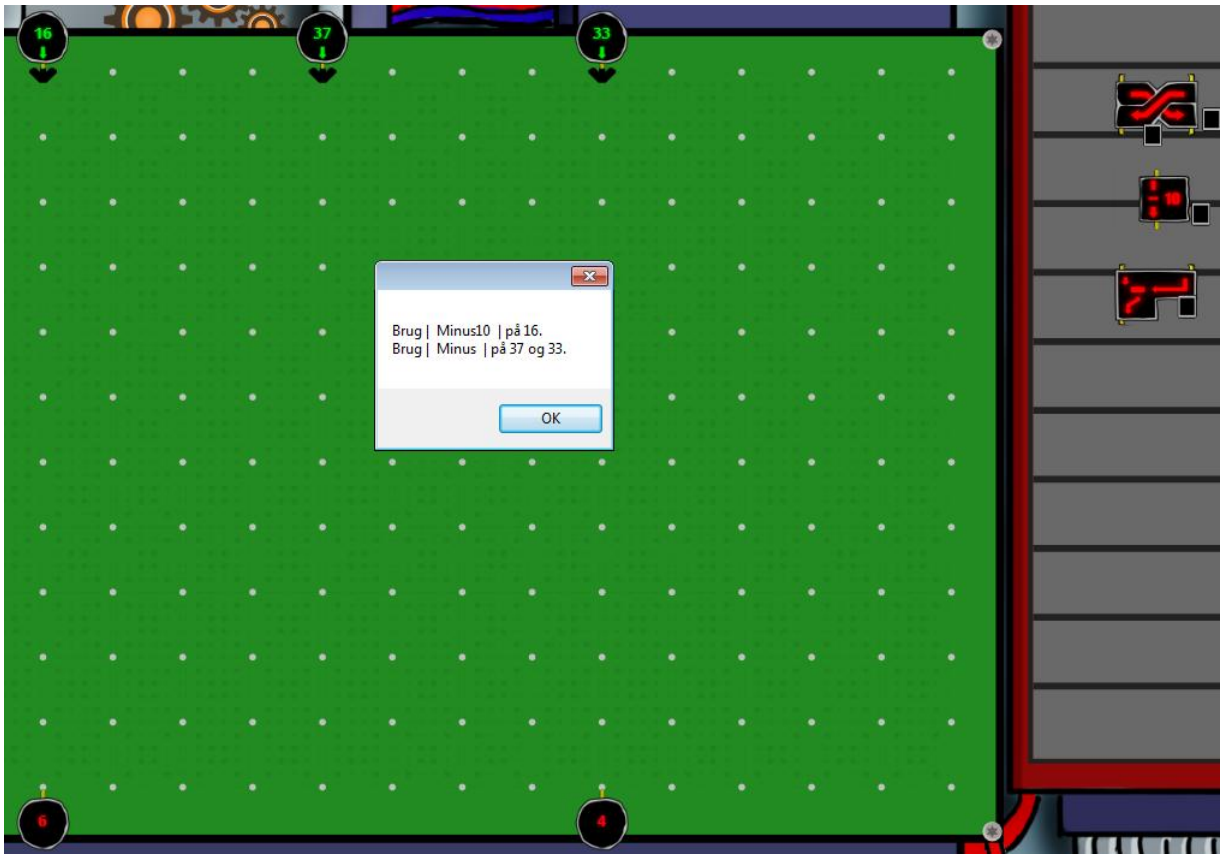


Figur 30 Alle baner i spillet er gennemført og robotten er nu glad og repareret frem for ked af det og i stykker

Hvis man forlader spillet og går til hovedmenuen, altså giver op, fra en tilfældigt genereret bane bliver man spurgt om man vil se løsningen på bane, da det kan være ærgerligt at give op på en bane og aldrig få chancen for at løse den igen. Figur 31 og Figur 32 demonstrere både en tilfældig genereret bane, samt funktionaliteten som giver løsningen på den hvis man giver op.



Figur 31 Popup som spørger om man vil se løsningen på den tilfældigt genereret bane



Figur 32 Visning af løsningen på en tilfældigt genereret bane

4.2 Spørgeskemaer

Spørgeskemaerne til undersøgelsen blev uddelt til alle der havde prøvet spillet. Spørgeskemaet bestod af fem meget simple spørgsmål. Testgruppen bestod af 41 elever, som blev bedt om at svare på spørgsmålene ud fra deres egen oplevelse med spillet.

Tre af spørgsmålene besvares med et tal mellem 1 og 10 og to af dem med ja, nej og måske.

I skemaet i Figur 33 er resultatet for de første 3 spørgsmål.

	1	2	3	4	5	6	7	8	9	10
Hvor svært?	1	1	3	3	10	7	9	3	3	1
Hvor sjovt?	3	0	2	1	4	1	9	9	5	7
Hvor nemt at lære?	3	2	5	1	5	6	1	8	7	3

Figur 33 Tabel over spørgeskema resultaterne

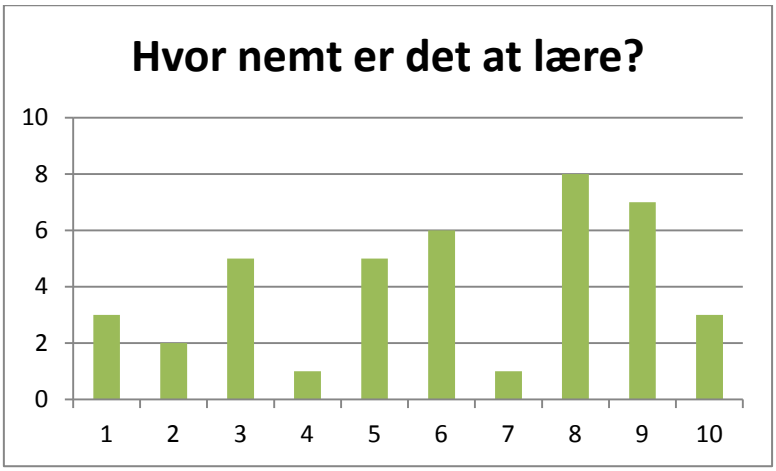
Følgende grafiske fremstillinger kan uddrages af tabellen i Figur 33.

Figur 34 Søjlediagram over spørgsmålet "Hvor svært er det?"

Af grafen på Figur 34 udledes det at gennemsnittet, der findes ved at dividere summen af svarene med antallet af svar, for sværhedsgraden af spillet ligger på 5,85, med en spredning, der for disse observationer ligger på 1,96.

Figur 35 Søjlediagram over spørgsmålet "Hvor sjovt er det?"

Af grafen i Figur 35 udledes det at gennemsnittet, for underholdningsværdien af spillet ligger på 7,05, med en spredning, der for disse observationer ligger på 2,54.



Figur 36 Søjlediagram over spørgsmålet "Hvor nemt er det at lære?"

Af grafen i Figur 36 udledes det at gennemsnittet, for hvor nemt eleverne synes det var at lære spillet ligger på 6,12, med en spredning, der for disse observationer ligger på 2,76.

Vi har brugt statistikprogrammet R, med funktionerne mean() og sd() til at udregne hhv. Gennemsnit og standardafvigelse.

Svarene på de to sidste spørgsmål findes i skemaet i Figur 37.

	nej	måske	ja
Ville du spille det hjemme?	11	8	23
Ville du spille det I matematiktimerne?	3	7	32

Figur 37 Skema over hvor brugeren ville spille det

For at visualisere dataene vises de her i to cirkeldiagrammer, hhv. Figur 38 og Figur 39.

Figur 38 Cirkeldiagram over om de ville spille det derhjemme

Figur 39 Cirkeldiagram over om de ville spille det i matematiktimerne

4.3 Lærer feedback

Ved testene af vores spil var Erik Ottar Jensen, som har været kunden for projektet, til stede og i dette afsnit vil summere hans feedback og forbedrings forslag. Den fulde mail fra ham kan ses i bilag 3. Vi har udeladt feedback som er resultatet af små bugs, såsom at komponentforbindelser en gang imellem har opført sig lidt underligt.

4.3.1 Generel feedback

Eriks generelle oplevelse af spillet var rigtig god.

"Det var super fedt at få spillet prøvet af og det er helt sikkert noget der er værd at arbejde videre med. Jeg ser mange spændende potentialer i det. " - Erik Ottar Jensen

Han var positivt overrasket over spillet og lagde særlig vægt på at det kørte godt og var bug frit. Han lagde også vægt på at eleverne efter hans mening også lærte noget af det, og at spillet engagerede og fangede dem, og at det var på trods af at pointsystemet ikke var så markant og velfungerende. Han fortsætter så med at sige at et pointsystem ikke er direkte er godt for læringsoplevelsen og at hvis spillet kan undvære det, hvilket vores kunne, ville det måske være en ide helt at fjerne det.

4.3.2 Forbedrings forslag

Forslagene er blevet kogt ned og inddelt i nogle kategorier:

1. **Ease of Use**
   1.1. Værktøjet til sletning af komponentforbindelser skal være nemmere at ramme med
   1.2. Tallene for et komponents output skal være større så det er nemmere at læse, samt at de matematiske operationer på komponenterne skal være større.
   1.3. Der skal være en restart knap som sletter alle komponenter og komponentforbindelser så det ikke skal gøres manuelt hvis man vil starte forfra uden at skulle igennem menuerne igen.
2. **Visuel feedback**
   2.1. Når banen vindes skal der gå strøm/gnist igennem systemet for at tydeliggøre man er lykkes og banen ikke slutter så brat.
   2.2. Når en ny bane bliver låst op skal det vises tydeligere så eleven bliver mere opmærksom på det
   2.3. Det skal være tydeligere hvilke baner er låste og hvilke ikke er.
3. **Indlevelse**
   3.1. Der skal være en intro video der tydeliggør spillets historie og giver en form for mål.
   3.2. når man vinder spillet skal der være en lille outro video hvor man kan se det hele endte godt.
   3.3. Som en ekstra belønning som gives gradvist når baner klares, skal man se robotten langsomt begynde at virke, f.eks. ved at den har nogle små animationer den udføre.
   3.4. Når banen vindes skal der gå strøm/gnist igennem systemet for at forstærke fantasien om at man faktisk arbejder på at reparere en robot.
4. **Gameplay**
   4.1. I Nogen baner ville det være gavnligt at begrænse antallet af et bestemt komponent man har til rådighed.

4.2. Eleverne ville måske synes det var sjovt hvis der var nogle ganske få baner med meget store tal, såsom millioner eller milliarder

4.3. Forslag til nye komponenter:

4.3.1. Gange komponent som ganger to tal, og ikke bare med 10

4.3.2. Komponent som sætter i anden potens

4.3.3. Komponent som sætter i tredje potens

4.3.4. Reintroducere dividere komponent, men denne gang med et ekstra output som har rest

4.4. Forbedring eller sletning af pointsystem

5. **Social**

5.1. Eleverne skal kunne lave baner til hinanden hvor de komponenter de kan stille til rådighed i deres egne baner skal låses op ved at klare bestemte baner i selve spillet.

5 Diskussion og konklusion

I dette afsnit tages resultaterne, som blev bearbejdet i resultatafsnittet, og bliver diskuteret for at forsøge at redegøre for, hvorfor den indsamlede data ser ud som den gør, hvilken effekt det har og hvad der kan gøres ved det.

Først evalueres vores observationer ved elevernes test af spillet, hvorefter lærerens feedback evalueres og til sidst diskuteres de resultater vi fik fra spørgeskemaundersøgelsen. Vi vil dernæst have et afsnit hvor vi reflektere over projektet og ser på mulige emner at arbejde videre med i fremtiden. Til sidst vil vi afsluttende konkludere på vores rapport for at runde projektet af.

5.1 Evaluering af elevtests

For spørgsmålet om hvor svært spillet var, lå svarene omkring midten af skalaen, lænende mod den svære ende, på 5,85 og en standardafvigelse på 1,96, hvilket betyder at 68% af svarene lå mellem 3,89 og 7,81. målet for et spils sværhedsgrad, og i særdeleshed for et spil, spilleren skal lære noget af, må være at ligge i midten af skalaen. Som tidligere nævnt er sværhedsgraden vigtig for spillerens fornøjelse med spillet. En sværhedsgrad der er for let gør spillet kedeligt og en svær sværhedsgrad kan gøre spillet frustrerende. En sværhedsgrad der ligger lige midt i mellem disse to poler tillader spillet at være udfordrende nok til at spilleren kun kan klare det hvis han tænker sig om, uden at det tager så lang tid at komme videre i spillet at det ender i frustration.

Spillets funktion er at tage et emne som spilleren har svært ved og forsøge at undervise i det uden at det føles som undervisning. Dette forsøger vi at gøre ved at stille opgaver op for spilleren som i starten er nemme at løse uden nogen dybtgående erfaring med positionssystemet, som vi har valgt at fokusere på her, og lidt efter lidt gøre eleven bevidst om hvordan det hænger sammen, ved at tilbyde det som et, nogen gange nødvendigt, værktøj til at løse et problem. Dette i sig selv bidrager til spillets sværhedsgrad og er ikke noget vi kan gøre så meget ved uden at miste læringsværdien. Noget der til gengæld kan gøres noget ved, som bidrager negativt til spillerens opfattelse af sværhedsgraden, er en stejl læringskurve. Hvis der ikke er nok baner til at blive fortrolig med spillets regler og spilbræt, eller hvis banernes kompleksitet stiger for brat, vil spillet blive opfattet som sværere. Ved at tilføje flere baner som mellemtrin, der hvor stigningen er højest glattes læringskurven ud og sandsynligheden for at spilleren har nået at lære alle de færdigheder, som den svære baner kræver af ham, er større.

Andre faktorer der kan influere sværhedsgraden kan være at spillet tillader negative tal. Da vi har med en masse frihed omkring hvordan man som spiller har lyst til at interagere med spillet på, har vi været nødt til at tillade at spilleren kan arbejde med negative tal. Det har vi været nødt til da vi ønskede at tillade et minus komponent, men ikke ville foretage nogen regneoperation der opførte sig anderledes end hvad den ville gøre uden for spillet, naturligvis for ikke at forvirre eleverne. De fleste elever på dette klassetrin har ifølge Erik Otter Jensen et eller andet begreb om negative tal, men det er ikke noget de beskæftiger sig ret meget med på det niveau. Derfor har vi ikke gjort det til et krav at bruge negative tal for at komme videre i spillet. Dog kan elever der vælger at prøve sig frem med negative tal finde nogle ret kreative løsninger på mange af banerne og forhåbentlig lærer de en smule om negative tal.

Spørgsmålet om hvor sjovt spillet er blev besvaret meget positivt af størstedelen af eleverne i testgruppen, hvilket afspejles i gennemsnittet for svarene, som ligger på 7,05, men med en større spredning på 2,54. Som nævnt før kan der være sammenhæng mellem dette og sværhedsgraden, men resultatet er et mål for hvor godt vi rammer med designet af vores features i spillet, hvor fængslende spilkonceptet er og vores evne til at skabe en god atmosfære. Det er selvfølgelig svært at lave et spil der appellerer til alle og styrken ved vores matematikspil er at de der har problemer med at lære matematik på mere traditionelle måder kan have stor glæde af at se den opstillet på alternativ vis. Når alt kommer til alt er et gennemsnit på 7,05, på trods af den store spredning, meget acceptabelt, men viser også at der er plads til forbedring.

Spørgsmålet om hvor nemt spillet var at lære, bliver besvaret meget uklart ud fra undersøgelsen. Med et gennemsnit på 6,12 lyder det meget balanceret, men med en spredning på 2,76, siger gennemsnittet meget lidt i forhold til hvad den generelle opfattelse har været. Det er selvfølgelig klart at man kan have meget forskellig grad af forståelse for matematik i en folkeskoleklasse, og der kan derudover have været stor forskel på hvorvidt tutorial delen af spillet har forklaret det grundigt nok til spillerne. Et andet problem, som Erik Ottar Jensen gjorde os opmærksomme på og også kan være skyld i at mange fandt det svært at komme i gang med spillet er, at tutorialdelen foregår i et stræk. Dvs. Du lærer om blandt andet om nogle af de mere avancerede komponenter og spilmekanikker langt før du kommer til at bruge dem. Ved at dele tutorialdelen op i mindre tutorials og give dem umiddelbart før spilleren møder den første bane der kræver kendskabet introduceres den avancerede del senere og spilleren får mulighed for at blive fortrolig med spillet inden da.

Til spørgsmålet om hvorvidt eleven ville spille det, hvis de kunne, hjemme, svarede forbløffende mange at de gerne ville. Det er naturligvis et tegn på at de havde det sjovt med spillet og at de ikke direkte forbinder det med skolearbejde. At det betyder at spillet på nuværende stadie kan konkurrere med de spil eleverne spiller på deres computer derhjemme er tvivlsomt, men at de er åbne for at tage det med ud af skolekontekst er positivt.

Til spørgsmålet om hvorvidt de gerne ville spille det i skoletiden svarede over 75 % af eleverne ja, hvilket kan tolkes som at de følte at det var en god afveksling fra normale matematiktimer.

Det er selvfølgelig farligt at lægge for meget værdi i en spørgeskemaundersøgelse, der er lavet på så lille et grundlag, men det giver alligevel et godt fingerpeg i de tilfælde hvor der kunne ses en tendens i svarene. Af mulige fejlkilder til undersøgelsen, kan nævnes at eleverne kan have haft problemer med at forstå spørgsmålene. Vi forsøgte at undgå dette ved at stille meget få og lette spørgsmål, desuden var vi til stede så de kunne spørge, hvis de var i tvivl om noget. Nogle af eleverne sad desuden i par og afprøvede spillet hvilket meget vel kan have influeret deres opfattelse af det.

Spørgeskemaundersøgelsen har sine fejl og mangler hvad angår, hvilke data vi bad om fra eleverne. Det ville være interessant at se ud fra en skemaundersøgelse om pigerne havde sværere ved spillet end drengene, fordi det ville give os en idé om hvordan spillet appellerer til hele målgruppen i stedet for kun en del af den.

Det ville også havde været interessant at se om der var nogen sammentræf mellem elevens klassetrin og deres oplevelse af spillet. Det ville have været meget nemt at tilføje, og eleven ville hurtigt kunne besvare det.

5.2 Evaluering af lærer feedback

Feedbacken fra Erik Ottar Jensen var meget konkret, og langt de fleste punkter er vi enige i, så i dette afsnit vil vi forklare hvordan vi vil løse de problemer som feedback viser, og for de punkter som vi ikke er enige i vil vi argumentere hvorfor.

Punkterne vi herunder referer til er de punkter som er listet i Lærer feedback afsnittet under Resultater.

Punkterne **1.1** og **1.2** kan løses på samme måde, nemlig ved at have færre punkter på spilbrættet, som man kan forbinde til. Dette vil gøre, at alting kan være større, da selve spilbrættet ville have samme størrelse, og der blot er kommet mere afstand mellem punkterne på brættet. Større komponenter har bedre plads til at vise hvad de gør på en visuel måde, og den større afstand mellem punkter vil tillade os at gøre komponentforbindelserne større, og derved nemmere at ramme. Hvis det så stadig viser sig at være for svært at ramme, kan vi kigge på andre muligheder, såsom at spilbrættet registrere hvis en museklik med sletningsværktøjet rammer den og så tjekker om det klik var i nærheden af en komponentforbindelse og i så fald videresende event'et til den komponentforbindelse.

Punkt **1.3** er triviel at klare, da alle vores objekter bliver håndteret i lister som vi til en hver tid kan tømme og genoprettes.

Punkt **2.1** og **3.4**, som jo er det samme men af forskellige grunde, er en af de ting som vi gerne ville have gjort allerede før testen, men som vi valgte at nedprioritere. Den kræver at vi har et billede af en elektrisk gnist som passer i vores grafiske stil hvilket var den begrænsende faktor, da selve det at bevæge den gennem systemet af komponentforbindelser og komponenter ville være lige til.

Punkt **2.2** ville kræve enden noget nyt grafik, eller nogle smarte animationer. Man kunne for eksempel når man vandt en bane zoom'e ud af robotten og så zoome ind på de baner man får låst op og så låsen forsvinde. Dette ville også kræve at det var grafisk tydeligere hvilke baner som var låste og hvilke som ikke var hvilket leder os til punkt 2.3.

Punkt **2.3** kunne løses på mange forskellige måder. Den simpleste, og måske også mest intuitive, ville være at have et billede af en låst hængelås over banen. Dette ville både fortælle brugeren meget klart at denne bane er låst, og det ville give os noget at fjerne som punkt 2.2 har brug for.

Punkterne **3.1** og **3.2** er lidt mere avanceret og meget tidskrævende. En eventuel løsning på dem kan ses i afsnit 5.3.2.2.4 Film.

Punkt **3.3** vil nok kræve et mere robust system til animation, da Silverlight ikke som sådan er tænkt til at animere så komplekse ting som en bevægelig robot. Dette ville dog kunne laves med de animationsfunktioner som Silverlight har, så løsningen ville nok være at lave en klasse, som kan bruges til at styre hver enkel del af robotten, og som er nem at bruge og implementere for koden udenfor.

Punkt **4.1** kan nemt laves. Spillet skal blot holde styr på hvor mange komponenter af en bestemt type er blevet brugt, og så fjerne det gældende komponent fra de komponent som er til rådighed, og tilføje den

igen hvis man ligger et af de eksisterende på plads. Dette skulle selvfølgelig også kommunikeres til spilleren ved f.eks. en nedtælling ved siden af komponenten som viser hvor mange af dem man har tilbage.

Punkt **4.2** er det punkt som vi ikke er enige i er en forbedring. Hvis så store tal skulle bruges i spillet skal de også vises, hvilket vi kunne gøre på to måder. Enten kunne vi gøre som var gør nu, altså blot skrive tallet. Det vil dog resultere i at når et tal som 1.000.000.000 bliver brugt så ville det enden fylde ekstremt meget og sandsynligvis dække for andre komponenter, eller tallet ville blive skrumpet ned til en størrelse hvor det ikke længere kunne læses. Den anden mulighed ville være at skrive f.eks., 1mil, 1mia. Vi tror ikke dette ville give den ønskede "wauw"-effekt fra eleverne, da det store i tallet er blevet gemt væk bag nogle bogstaver. Ingen af disse muligheder er tilfredsstillende og vi synes derfor ikke dette forslag ville forbedre spillet.

Punkt **4.3** kunne være interessant, men da det kræver en del arbejde at tilføje nye komponent typer til spillet med det nuværende system skulle man nok indføre det nye komponent system, som er diskuteret i afsnit 2.3.3 Komponenter, først.

Punkt **5.1** er et punkt som til en vis grad allerede er blevet omtalt i afsnit 3.3.1 Strukturen af levels, men er en feature som vi har nedprioriteret. Den ekstra detalje ved dette forslag som vi ikke selv havde talt om var at man gradvist fik låst op for brugen af de forskellige komponenter i ens egne baner, hvilket giver en god belønning løbende igennem spillet. Det tage i betragtning ville dette nok være noget vi ville have prioriteret højere. Takket være vores standardiseret *LevelSpecification* ville det være nemt at implementere muligheden. Det kræver blot en menu til at udfylde de relevante ting i.

5.3 Refleksion

I refleksionsafsnittet tager vi og kigger på hvad vi ville gøre anderledes hvis vi kunne lave projektet forfra, samt hvad man burde fortsætte på for at bringe projektet videre fra det stadie det befinder sig i nu.

5.3.1 Hvis vi skulle gøre det hele igen

I dette afsnit vil vi kigge på vores projekt og med den viden og erfaring vi har erhvervet os igennem forløbet og vil vi forklare hvad vi ville have gjort anderledes, hvorfor og hvordan.

5.3.1.1 Komponent system

Som beskrevet i afsnit afsnit 3.2 Komponenter kom vi på en bedre struktur til vores komponenter, som var meget mere skalerbar og krævede meget færre specialchecks andet sted i programmet. Denne struktur af komponenter ville have gjort meget andet funktionalitet nemmere, hurtigere og bugfrit at lave, og selve strukturen havde ikke taget væsentligt længere tid at lave. Det er derfor en, på alle måder, bedre løsning end vores nuværende og ville have givet os mere tid til at nå nogle at lave nogle af de features vi har endt med at nedprioritere.

5.3.1.2 Database kommunikation

Databasekommunikationsdelen af programmet er et af de steder, der desværre ikke er blevet helt så færdigudviklet som det burde have været. Selve designet af databasen er tilstrækkelig til det behov vores spil opstiller, men sikkerheden af transaktionen, samt hastigheden er noget der kunne være arbejdet på i langt højere grad.

De sikkerhedsmæssige risici, som er forbundet med vores metode til at sende data til php scriptet og de data som php scriptet sender tilbage ville være uacceptable i et færdigt program. Som de ser ud nu opfylder de deres primære funktion, men ikke på en hensigtsmæssig måde.

Programmet er sårbart f.eks. overfor metoder som SQL injektioner. Dette er en hacking metode, der gør det muligt at ændre den forespørgsel, som hackeren formoder den indtastede data bliver brugt til.

Som nævnt tidligere er der forskellige metoder, hvorpå en større grad af sikkerhed kunne opnås, men vigtigst af alt er at kryptere den data, der sendes til og fra php scriptet. Det skal være noget, der gør at udelukkende anmodninger, der foretages gennem programmet kan blive processeret.

5.3.1.3 bedre adskillelse af MainPage og Spilbrættet

I vores design af spillet blev MainPage klassen brugt både som spilbræt og som rodobjekt for grafikken. Dette gav mening i starten da det eneste i vores program var spilbrættet og programmet voksede udenom dette. Problemet blev at da programmet voksede og MainPage begyndte at være rodobjekt for en masse objekter, der ikke havde noget med spilbrættet at gøre, og det blev meget uklart hvad MainPage klassen repræsenter i programmet.

En klar forbedring ville være at holde MainPage som rodobjekt med adskille det fra spilbrættet, hvilket ville give en langt renere kode, der var nemmere at forstå, og gøre det nemmere at vedligeholde den i fremtiden.

5.3.1.4 Testing

Et problem som vi har indset senere i spillets udvikling er at spillet er bygget på en måde som gør det svært at teste visse funktionaliteter helt igennem, og vi har derfor brugt mere tid på at finde og præcisere bugs end vi har været tilfredse med. En måde at løse dette på var at vi fra starten havde sørget for at koden var bygget så den kunne testes ordentligt og systematisk. Når det var gjort kunne vi have draget nyttet af Unit Testing som, på trods af en smule overhead ved at skrive selve tests, utvivlsomt ville have spare os mange timer, og til slut have givet et generelt bedre produkt.

5.3.2 Fremtidige udsigter

5.3.2.1 kortsigtede forbedringer

De fleste af de forbedringer som Erik Ottar Jensen forslog var rigtig gode og er nogle vi ville kunne lave hurtigt og nemt, og som vi derfor ville prioritere at få gjort. Undtagelserne er intro og outro videoer, samt animation af robotten så den kan bevæge sig.

Videoerne ville være en meget tidskrævende proces, men ideen er god så man kan læse mere om den i afsnit 5.3.3.2 Langsigtede forbedringer. Animation af robotten er ligeledes god, og den ville nok tage markant kortere tid, og ville nok være noget vi ville prioriteret at få gjort, da spillet ville gavnes af at have noget mere liv.

Resten af forslagene blev diskuteret og forklaret hvordan de kunne føres ud i livet i afsnit 5.2 Evaluering af lærer feedback.

5.3.3.2 Langsigtede forbedringer

5.3.3.2.1 Mere intelligente komponentforbindelser

Som omtalt i afsnit 2.3.4 Komponentforbindelser havde vi på et tidspunkt arbejdet på at gøre komponentforbindelserne mere intelligente, så de selv kunne finde vej fra et hvilket som helst punkt til et hvilket som helst andet punkt, hvis en vej fandtes. Som vi fandt ud af så ville dette kræve en rigtig søgealgoritme, som så blev kørt hver gang brugeren slap en komponentforbindelse. A* algoritmen er som sagt et oplagt valg, da den er meget velegnet til rutefindings problemer, er effektiv og med en ordentlig heuristik, såsom korteste vej i fugleflugt, garantere os den optimale vej. Selve algoritmen A* er ikke ekstrem avanceret, men nogle af de datastrukturer vi ville få brug for ville kræve en del tid, og derfor ligger denne opgave under langsigtede forbedringer.

5.3.3.2.2 Farve-kodet tal

Featuren som er beskrevet udførligt i afsnit 2.1 Spilkoncept er noget vi ville arbejde på i fremtiden, hvis projektet skulle fortsætte. Det ville dog tage lang tid, for det første at udtænke det på en måde der ville virke intuitiv for spilleren, men ville også kræve en omfattende omstrukturering og omskrivning af størstedelen af programmet.

Problemet ville være at alle tal, som nu bliver håndteret som integers ville være nødt til, at blive tilknyttet en yderligere værdi i form af separate cifre, hver med en farve. Dette nye objekt skal håndteres af komponenterne i stedet for integers og skulle manipuleres både de individuelle dele af tallene og farvene, alt efter hvad vi beslutter omkring dem.

5.3.3.2.3 Bedre algoritme til automatisk generering af baner

Som nævnt i slutningen af afsnit 2.3.5 Level system er det ikke ønskeligt at gå over til kun automatisk genereret baner. Men det ville stadig være godt hvis man kunne lade automatisk genereret baner med mange løsnings muligheder have en større rolle i spillet.

Vi havde jo som sagt i afsnit 2.3.5 Level system arbejdet på en algoritme til dette, men den endte med at være for grundig. Grund ideen er dog god nok men problemet er bare at søgerummet er alt for stort. En løsning til dette generelle problem er ofte at gøre algoritmen intelligent, f.eks. så den ikke leder i veje som ikke ser lovende ud. Ved at fokusere på de veje som ser ud til at give mange løsninger og så bare terminere når f.eks. et bestemt sæt af spiloutputs er fundet som kan opnås på 6 forskellige måder, vil vi skære søgerummet markant ned. Udfordringen ved dette er at finde en heuristik som kan bestemme hvilke veje ser lovende ud, og det er ikke noget vi lige umiddelbart ved hvordan vi skal løse, hvilket resultere i at dette ligger under langsigtede forbedringer. Men når først det problem er løst, hvilket som sagt også er det sværeste, kan vi begynde at fin tune algoritmen så vi fravælger løsninger som ikke er interessante. f.eks. er input 2, 3, 4 med output 20,30,40 ikke interessant hvis man har gange 10 komponenten, også selvom der er rigtig mange andre måder at løse problemet på.

5.3.3.2.4 Film

De fleste spil indeholder på en eller anden måde et stykke film, ofte flere i løbet af spillet, der driver historien og fastholder spillerens fokus på det overordnede mål og bruges i starten som incitament, da den gerne skal vise det langsigtede perspektiv med spillets historie, samt hvorfor spilleren skal føle noget overfor problemet som er præsenteret i spillets historie.

Problemet med at have en film i vores spil er at det er meget tidskrævende at lave den slags animerede film, og er samtidig noget vi ingen form for erfaring har med. Et alternativ kunne være at bruge de billeder vi allerede har i spillet, og så f.eks. animere arme, ben og hoved på robotten og så lave en "film" på den måde. Problemet er så med animationer og silverlight er at, hvis de skal kodes ind i programmet bliver det meget rodet, og kvaliteten af resultatet kunne være tvivlsomt, specielt den arbejdsbyrde taget i betragtning det ville være at lave på denne måde. En anden mulighed er at lave animationen på samme måde, men i et program der er beregnet til det, så som Adobe Flash, kompilere det til en videofil og lægge den ind i

programmet og så bare afspille det, men det er ikke noget vi har den store erfaring med, og dette kan derfor også tage meget lang tid.

5.4 Konklusion

Vi har i projektet udviklet et computerspil til læring af matematik, som gemmer matematikken væk fra brugeren, og præsenterer det som et værktøj mere end som et mål. Formålet med det har været at få eleverne til at glemme at de faktisk sidder og laver matematik, noget som nogle ikke synes er så interessant, men stadig få dem til at bruge matematikken. Dette mener vi selv, og bakkes op af de feltundersøgelser vi foretog, er lykkes. Ikke dermed sagt at vi har et perfekt produkt, da der helt sikkert er plads til forbedringer, men vi har bestemt fat i noget om er værd at arbejde videre med.

Vores arbejde med specielt positionssystemet blev ikke som vi først havde forventet. Vi lavede nogle ændringer igennem projektet i forhold til den originale ide med farvekodede tal, hvilket måske har været en god ting, på trods af at noget af fokussen på positionssystemet er gået tabt derved. Det ekstra lag af kompleksitet i spillet ville nok have været for meget, da der allerede nu mente at spillet var svært at lærer. Det er vigtigere at brugerne bruger tiden på at lærer matematik end at lære hvordan vores spil virker. Det er dog ikke dermed sagt at vi ikke har givet eleverne mulighed for at lærer mere om positionssystemet og hvordan det kan bruges. Filter komponenterne giver rig mulighed for manipulering af tal i forhold til positionssystemet, og gange 10 gør lidt det samme. Der var en bane i vores test som kun brugte filter komponenter, og det var rigtig interessant at se eleverne tænke over hvordan de kunne løse dem, og se "aha!"-glæden da de løste det.

I forhold til at give læren en form for information omkring elevernes evner, styrker og svagheder har vi lavet en del grundarbejde, men det var faktisk ikke så ligetil som vi først antog at bruge det data som spillet stiller til rådighed til at danne sig et overblik over en elevs evner. Vi endte med at kigge på den tid en elev bruger på at klare en bane, og vise det i en graf hvor X er antallet er baner som er klaret, og Y er tiden det har taget. Man kan så blotte hver enkel sektion for sig selv, så man for eksempel kan se om der er en bestemt sektion som tager længere tid end resten. Dette er relevant da hver sektion har nogle bestemte komponenter den bruger og man på den måde kan se om der er nogle komponenter, og derfor matematiske operationer, som eleven har svært ved at bruge. Derudover indsamler databasen også informationer om hvilke komponenter som der er brugt og ikke er brugt i en bane, og dette kan ligeledes give læren en ide om en elev måske bevidst eller ubevidst undgå visse matematiske funktioner.

For at sikre at spillet kan spilles af både de dygtigste og de svageste har vi designet det så man hurtigt kan få adgang til mange forskellige baner, og man ikke ender med at sidde fast i et bestemt bane og må stoppe helt med at spille spillet. Vi har samtidig nogle meget sværere baner som kan give de bedste kamp til stregen, men de er placeret så de ikke spærre for at dem som ikke kan klare dem ikke kan komme videre. Til sidst har vi også lavet muligheden for at spillet generere en ny bane til spilleren i tre forskellige sværhedsgrader som spilleren selv kan vælge imellem. Så selv hvis brugeren skulle sidde fast et sted kan de altid vælge at træne lidt i nogle af de automatisk genereret baner.

Rent software-mæssigt mener vi at vi har lavet et godt, robust produkt som båder er skalerbart og relativt nemt at vedligeholde og udvikle nye features til. Som diskuteret andet steds i rapporten er der forbedringer vi kunne foretage, men vi er bevidst omkring dem, og har forslået løsninger til dem.

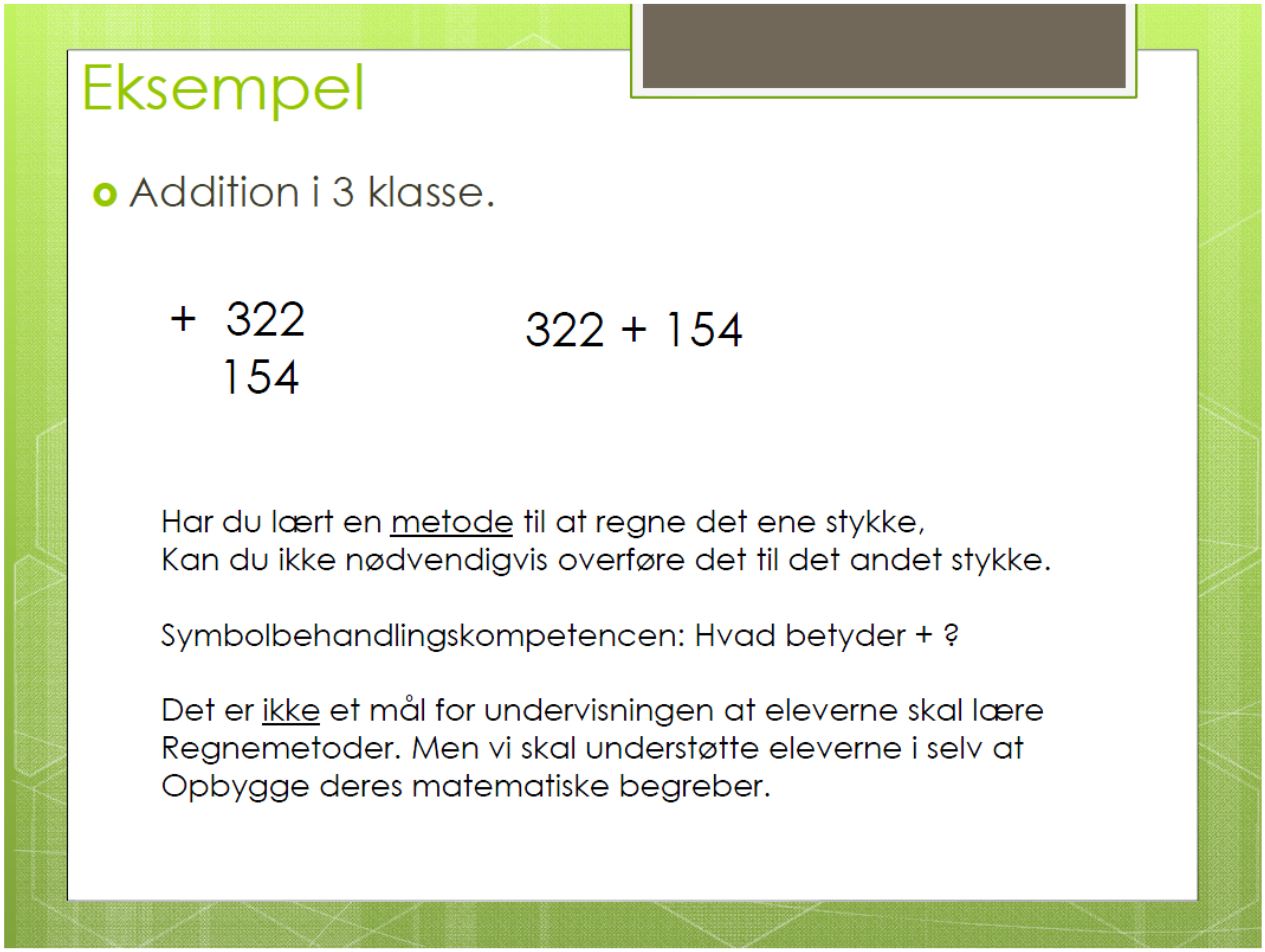
Til slut vil vi sige at det har været et rigtig interessant og spændende projekt at arbejde på, som har givet os både faglige udfordringer og udfordringer om hvordan vi kunne designe et produkt som er intuitivt at bruge og nemt at lære til en målgruppe som er meget forskellige fra os selv, eller hvad man normalt bliver præsenteret for i uddannelsesforløbet. Projektet har resulteret i en velfungerende produkt som opfylder, omend ikke perfekt men stadig rigtig godt, de målsætninger og krav som blev stillet til det, og som med en smule mere arbejde bestemt kunne blive til noget brugbart for de danske folkeskoler.

Referencer

1. http://ing.dk/artikel/folkeskolen-taber-matematik-talenter-115319
2. http://msdn.microsoft.com/en-us/magazine/dd419663.aspx
3. Artificial Intelligence A modern Approach, 3rd international edition, Stuart Russel & Peter Norvig, page 93
4. https://nuget.org/packages/Microsoft.Bcl.Async
5. http://php.net/manual/en/function.mysql-connect.php
6. http://php.net/manual/en/mysqli.query.php
7. http://msdn.microsoft.com/en-us/library/vstudio/ms752347(v=vs.90).aspx#binding_to_collections

Bilag

Bilag 1



Bilag 2

```
public class GenericComponentModel
    {
        public int InLimit { get; private set; }
        private int _outLimit = 1;
        public int OutLimit { get { return _outLimit; } private set { _outLimit = value;} }
        public MainPage.ComponentSizeType SizeType { get { return OutLimit == 1 &&
             InLimit == 1 ? MainPage.ComponentSizeType.Small :
             MainPage.ComponentSizeType.Large; } }
        public int LeftMargin { get; private set; }
        public int RightMargin { get; private set; }

        public List<Func<int, int, int>> ChosenMathFun { get; private set; }
        public List<Uri> ImageSources { get; private set; }


        public GenericComponentModel(List<Func<int, int, int>> mathfunsies, List<Uri>
             imageSources, int inLimit, int leftMargin)
        {
            ChosenMathFun = mathfunsies;
            ImageSources = imageSources;
            InLimit = inLimit;
            LeftMargin = leftMargin;
        }
        public GenericComponentModel(List<Func<int, int, int>> mathfunsies, List<Uri>
             imageSources, int inLimit, int leftMargin, int outLimit, int rightMargin)
             : this(mathfunsies, imageSources, inLimit, leftMargin)
        {
            OutLimit = outLimit;
            RightMargin = rightMargin;
        }
    }

    public static class MathFunctions
    {
        public static int Plus(int input1, int input2)
        {
            return input1 + input2;
        }
        public static int Minus(int input1, int input2)
        {
            return input1 - input2;
        }

        private static int Filtero1(int input1, int filter)
        {
            int output = ((int)(input1 / filter) % 10) * filter;
            return output;
        }
        private static int Filtero2(int input1, int filter)
        {
            return input1 - Filtero1(input1, filter);
        }

        public static int Filter1o1(int input1, int input2 = 0)
        {
            return Filtero1(input1, 1);
        }
        public static int Filter1o2(int input1, int input2 = 0)
```

```
    {
        return Filtero2(input1, 1);
    }
    public static int Filter10o1(int input1, int input2 = 0)
    {
        return Filtero1(input1, 10);
    }
    public static int Filter10o2(int input1, int input2 = 0)
    {
        return Filtero2(input1, 10);
    }
    public static int Filter100o1(int input1, int input2 = 0)
    {
        return Filtero1(input1, 100);
    }
    public static int Filter100o2(int input1, int input2 = 0)
    {
        return Filtero2(input1, 100);
    }
}
```

Bilag 3

Mail fra Erik Ottar Jensen med hans feedback baseret på egen mening samt observationer af eleverne:

"Jeg ville bare sige tak for i dag. Det var super fedt at få spillet prøvet af og det er helt sikkert noget der er værd at arbejde videre med.

Jeg ser mange spændende potentialer i det. Nu skriver jeg lige nogen af mine overvejelser så jeg ikke glemmer dem. Jeg ved godt i har travlt med jeres rapport nu så I ikke får tid til at lave noget på spillet.

Interface:
- Der skal gøres et eller andet så det bliver lidt nemmere at få spillet til at gøre det man vil. Hammeren er lidt for præcis, det er svært at ramme stregerne med den. og stregerne trækker lidt underligt en gang i mellem.
- Der skal være en "reset level knap" så man ikke skal slette alt individuelt hvis man skal begynde forfra.
- ang. levels så skal det være mere tydeligt hvilke levels man har gennemført og hvilke levels der er mulige at vælge.
- Som der var en af Jer der sagde til mig i dag så skal tallene gøres større så det bliver mere tydeligt hvad der står.
- Når den sidste del af linjerne forbindes skal der være noget der viser at der går strøm igennem eller sådan noget lignende.
- I får alligevel ikke specielt mange elever til at læse den hjælpetekst i har skrevet så måske skal den del af instruktionen tænkes anderledes. KÆMPE pile der blinker eller noget. Ja KÆMPE, for eleverne er i gang med et spil og hvis der er nogen der begynder at fortælle dem hvad de skal gøre så prøver de allerførst at springe det skridt over og prøve selv.

Gameplay:
- Jeg tænker om det i nogle baner vil være en ide at begrænse antallet af komponenter man kan bruge. fx. minus en og plus en. Jeg ved ikke helt om det er godt eller skidt at man kan vælge dem ubegrænset.
- Jeg savner en gange komponent, præcis som plus og minus komponenterne.
- Hvis vi laver nogle udvidelser af baner kunne det være interessant at have nogle baner med megastore tal. millioner og milliarder osv. Det vil der være nogle elever der syntes er megafedt.
- Hvad med en komponent der sætter i anden og tredje.
- Og så skal vi klart have en divisions komponent der dividere med en rest så der kommer to udgange ud af den.

Social: vi skal kunne gøre sådan at eleverne kan lave baner til hinanden. Måske ville det være en ide at man ved at spille spillet får låst komponenter op således at man kan bruge forskellige komponenter gradvist. fx. når man har spillet 3 levels så kan man lave en bane hvor man har en plus komponent. osv. Hvis vi kan få en sådan del til at fungere så er det her det største potentiale ligger i spillet. Tror jeg.

Historie: Jeg ved ikke om det er muligt at lave en introdutionsvideo til det nemt i et eller andet program, men jeg tror at en eller anden form for historie der følger en igennem levelsne vil være med til at kunne fastholde eleverne i længere tid ved de opgaver de har svært ved. Det er der i hvert fald noget der tyder på fra andre spil. Og så en lille animation hver gang man laver en del af robotten færdig. Og selvfølgelig slut scenen hvor man går hånd i hånd med sin robot ned af gaden i solnedgangen. :-)

Observationer:
Nå men ellers er jeg positivt overrasket over spillet. Og jeg syntes især det er værd at bemærke at det fungere superfint. Det engagere eleverne og jeg tror at de lærer noget af det, uden et velfungerende pointsystem. Det syntes jeg er rigtigt interessant. Især fordi pointsystemer i sig selv ikke nødvendigvis er noget godt at tilføre en læringsoplevelse. Så det kan være at vi skal overveje rigtig grundigt hvordan det point system skal skrues sammen og om det overhovedet skal være et. Jeg tror at vi kan bruge point systemet til nogle ting i fremtiden men jeg er ikke helt sikker på at det er nødvendigt. Især ikke hvis man kan se at robotten begynder at bevæge sig jo længere man kommer frem. "