Semesterprojekt World of Zuul- Fire Escape

Universitet:	Syddansk Universitet Odense
Fakultet:	Det Tekniske Fakultet
Uddannelse:	Civilingeniør i Software Engineering
Kursusnavn:	SI1-OOP Semesterprojekt
Afleveringstype:	Projektrapport
Projekttitel:	Fire Escape
Projektgruppe:	Gruppe 02

Projektforfattere:

Morten Krogh Jensen	*mortj18@student.sdu.dk*
Jacob Wowk Jørgensen	*jacjo18@student.sdu.dk*
Thomas Steenfeldt Laursen	*tlaur18@student.sdu.dk*
Alexander Nguyen	*alngu18@student.sdu.dk*

Projektvejledere:

Lone Borgersen	*lobo@mmmi.sdu.dk*

Dato:

21-12-2018

2. Resumé

2.1 Problem

Rapportens primære problemstillinger bygger på børns utilstrækkelige viden om brandsikkerhed. Falck ved ikke hvordan de skal formidle disse seriøse informationer til børn. Børn forstår ikke hvad de skal stille op, så man ser at børn gemmer sig i stedet for at finde ud i sikkerhed. Denne utilstrækkelige viden gælder ikke udelukkende børn, men også forældrene. Forældrene ved ikke nok om brandsikkerhed til at kunne videregive vigtige informationer til deres børn.

2.2 Metode

Problemet blev bygget op på et teoretisk problem. Gruppen brugte forskellige informationssøgningsmetoder til at finde frem til relevant information som skulle danne grundlag for projektet. Gruppen brugte interviews med ressourcepersoner som en vej til at indhente data om problemets omfang.
Projektet har en praktisk løsning hvilket betyder, et produkt er direkte involveret i løsningen af problemet. Projektet blev bygget op af en 1. iteration og en 2. iteration, som består af henholdsvis en kommandobaseret brugerflade og en grafisk brugerflade.

2.3 Løsning

Der blev fokuseret på at lære de nyere generationer om brandsikkerhed for at de kunne tage hjem og starte en diskussion i hjemmet. Gruppen har derfor valgt at konstruere et spil til at fange den unge målgruppe og formidle information om brandsikkerhed.

Spillet er bygget op fra World of Zuul-platformen, som gruppen har udvidet med funktionalitet fra både dem selv og projektoplægget. Spillet går ud på, at man skal slippe ud fra en brændende bygning, ved hjælp af løsningsorienteret tænkning. Brugeren vil i en gennemspilning blive undervist i brandprocedurer og diverse relevante informationer.

2.4 Resultat

Efter 1. iteration havde gruppen fremstillet et tekstbaseret program som ved hjælp af World of Zuul-platformen var i stand til at udspille et scenarie hvor man befinder sig i en brændende bygning og skal slippe ud. Efter 2. iteration fik programmet tilføjet en grafisk brugergrænseflade og yderligere funktionalitet. Det endelige program opfyldte alle krav, hvoraf to af dem var delvist opfyldt. Det lykkedes gruppen at fremstille et program der underholdende kan undervise brugerne i brandsikkerhed ud fra en World of Zuul-platform.

3. Forord

Rapporten er en del af et projekt på 1.semester. Den har til formål at fremlægge gruppens bud på en applikation bygget på spillet World of Zuul. Formålet var at styrke gruppens evner inden for problemorienteret projektarbejde, samt at kombinere vores læring fra Introduktion til software Engineering og objektorienteret programmering til et produkt som opfylder de meste gængse regler for et program. Rapporten henvender sig primært til softwareingeniører og lærer, eller andre personer med den nødvendige baggrund inden for software. Gruppen har i løbet af projektets udførelse fået hjælp af en række person som gruppen derfor gerne vil takke.

- **Frederik Verner Helth og hans gruppe** – En anden gruppe som hjalp med udgivelsestesten, og sørgede for konstruktiv feedback som kunne bruges til vurdering af elementer i spillet.

- **Henrik Steinmann Heidelbach** - Tidligere spiltester hos IO Interactive, som hjalp gruppen med en brugertest, der med sin erfaring i spiltest gav konstruktive forslag til spillets funktioner.

- **Lone Borgersen** – Gruppens vejleder som igennem hele forløbet var klar til at guide gruppen, og svare på de spørgsmål gruppen måtte have.

- **Mathias Lilleskov Adler Jensen** – Et medlem af gruppen som var med i 1.iteration, men som desværre ikke afsluttede semesteret sammen med gruppen.

- **Morten Andersen** – Chef for forebyggende afdeling i Lolland-Falster brandvæsen, som hjalp gruppen med at få et realistisk syn på problemets omfang.

_____ _____

Morten Krogh Jensen Thomas Steenfeldt Laursen

_____ _____

Jacob Wowk Jørgensen Alexander Nguyen

Rapporten er afleveret d. 21/12-2018

4. Indhold

I. Produktrapport

II. Procesevaluering

5. Læsevejledning

Rapporten beskriver gruppe 02's semesterprojektet på 1. semester. For at få mest ud af den, foreslås det, at den læses kronologisk. Igennem rapporten beskrives både første og anden iteration af programmet. Hvert delafsnit i både rapportens hoveddel og diskussionsdel er opdelt i to dele; Første iteration og Anden iteration. Dog med undtagelse af enkelte afsnit, hvor beskrivelsen af de to iterationer forekommer mere sammenhængende. Hvis man igennem rapporten bliver i tvivl om et ords betydning, kan man med fordel kigge i rapportens ordliste. Igennem rapporten støder man på billeder hvor indholdet kan være småt. Disse billeder vil også være at finde i Bilaget. Rapporten henvender sig primært til softwareingeniører og lærer, eller andre personer med den nødvendige baggrund inden for software.

6. Redaktionelt

Alle i gruppen er blevet enige om at alle har ansvaret for rapporten, da gruppen kigger alt der er skrevet igennem og gennemgår det med hinanden og tjekker om der skal tilføjes mere. Gruppens medlemmer har forskellige hovedansvarsområder, som det fremgår af skemaet herunder:

	Morten Jensen	**Jacob Wowk Jørgensen**	**Thomas Steenfeldt Laursen**	**Alexander Nguyen**
ANSVAR FOR	- Resume - Forord - Indledning: Interesseparter - Indledning: Projektrammerne - Indledning: Metoder - Hovedtekst: Spillets koncept - Hovedtekst: Implementering - Procesrapport	- Indledning: Problemets sammenhæng og kompleksitet - Indledning: Tidsplan - Hovedtekst: Spillets koncept - Hovedtekst: Krav	- Læsevejleding - Redaktionelt - Indledning: Dokumentation - Indledning: Problemets historie og baggrund - Indledning: Problemformulering. - Hovedtekst: Spillets koncept - Hovedtekst: Design - Diskussion - Konklusion - Oversigt over kildekode - Brugervejledning	- Ordliste - Hovedtekst: Gamification - Hovedtekst: Spillets koncept - Hovedtekst: Implementering - Hovedtekst: Test - Perspektivering - Projektlog
BIDRAGET TIL	- Redaktionelt - Ordliste - Indledning: Problemformulering. - Hovedtekst: Krav - Hovedtekst: Test - Perspektivering	- Resume - Redaktionelt - Ordliste - Indledning: Problemformulering. - Indledning: Metoder - Hovedtekst: Gamification - Hovedtekst: Design - Hovedtekst: Test - Procesrapport	- Resume - Redaktionelt - Ordliste - Indledning: Problemets sammenhæng og kompleksitet - Indledning: Projektrammerne - Hovedtekst: Krav - Procesrapport	- Redaktionelt - Ordliste - Indledning: Problemformulering. - Indledning: Metoder - Hovedtekst: Krav - Procesrapport

7. Ordliste

Nedenstående liste beskriver en række ord der bliver nævnt rundt om i rapporten.

7.1 Prototype

NPC

NPC er en forkortelse for Non Player Character som er en karakter der ikke er styret af en spiller, men som spilleren kan interagere med.

Items

Spillet består af forskellige genstande som spilleren er i stand til at samle op. I denne rapport anvendes ordet Items når de omtales.

Kommando

Ordet kommando bruges til at beskrive det brugerinput spillet får som skal få programmet til at udføre diverse handlinger

Highscore

Highscore er en samling af de ti højeste point opnået af brugere.

Brugeren/spilleren

Når brugeren/spilleren bliver nævnt, så henvises det til målgruppen som er blandt andet unge.

7.2 IT

JAVA

JAVA er et objektorienteret programmeringssprog som bliver benyttet i hele dette projektforløb.

FXML

FXML er en XML-baseret grænseflade der bruges til at definere scene graph for JavaFX.

Scene graph

En scene graph er en samling af de nodes som vises på skærmen igennem programvinduet.

Node

Enhed/komponent som vises på skærmen igennem programvinduet. F.eks. en knap. Kan også være en parent.

Parent

Node som kan indeholde andre nodes. F.eks. borderpane og gridpane.

JAR-fil

JAR-fil er en forkortelse for JAVA arkivfil som er en speciel fil der bruges af JAVA. Denne fil indeholder alle Java klasser, samt andre filer krævet for at programmet kan køre.

If-statement

Bruges til at bestemme den kode der eksekveres baseret på en betingelse som enten er sand eller falsk.

For-loop

En for-loop eller for-løkke er en bestemt type løkke hvor en tællevariabel oprettes i løkkens initialisering. Især god når man ved præcist hvor mange gange man ønsker et kodestykke gentaget.

Metode

En metode indeholder kodestykker. Ved at kalde metoder, kan man eksekvere det samme kodestykke flere gange uden at skulle skrive det igen.

ArrayList

En ArrayList er en datatype som kan indeholde en mængde instanser af andre datatyper. Mere fleksibelt end almindelige arrays. Indholdet er ordnet og har hvert et indeks.

Presentation, Domain, Data Access

Et system er normalt delt op i tre forskellige pakker; præsentation, domæne og data. Præsentationslaget står for al interaktion med programmets bruger, både input og output. Domænelaget indeholder programmets funktionalitet og foretager alle de logiske operationer. Datalaget står for at give programmet afgang til skrivning til og læsning fra filer.

CLI

CLI er en forkortelse for Command-Line Interface og er en programgrænseflade hvor brugeren indtaster kommandoer i form af tekst. Programmets output er også i form af tekst.

GUI

GUI er en forkortelse for Graphical User Interface og er en programgrænsefalde hvor brugeren interagerer med et program gennem grafiske objekter og billeder.

Controller

Controller er en Java fil som laver reference til XML-dokumentet som har informationen om brugergrænsefladens udseende. Controlleren hjælper udvikleren med at separere brugergrænsefladens logik fra den resterende kode.

EventHandler

En metode som køres når brugeren laver en bestemt handling. F.eks. et klik på en knap.

8. Indledning

I starten af projektet foretog gruppen en brainstorm over problemer som de kunne arbejde med i projektet. Brainstormen endte med, at der i dette projekt fokuseres på følgende hovedproblem:

For mange mennesker er ofre i brandulykker

I denne indledning undersøges det hvordan dette problem er relevant i nutidens samfund, hvordan det er opstået, hvem interesseparterne er, og der foretages en afgrænsning af projektet. Derudover præsenteres projektets problemformulering.

8.1 Introduktion

Dokumentation af problemet

Hvert år forekommer brandulykker som forvolder skader for flere millioner af kroner og i de værste tilfælde skader eller endda dræber menneskerne indblandet. Ifølge beredskabsstyrelsen er der i de seneste ti år omkommet 70 mennesker fordelt på 68 brande i gennemsnit pr. år i Danmark[1]. Ydermere, er antallet af omkomne i brand steget med hele 20 % i Danmark fra år 2016 til 2017[2].

Problemets historie og baggrund

Ifølge en undersøgelse foretaget af ingeniørforeningen IDA, snakker for få forældre med deres børn om brandsikkerhed[3], dette gør problemet stadig er aktuelt. Kun 42 procent af de adspurgte havde snakket med deres familie/børn om hvordan man skulle forholde sig i tilfældet af brand. Langt fra alle har informeret deres børn om hvad de skal gøre hvis en brand bryder ud, dette vil resultere i, at børn ikke er i stand til at redde dem selv, hvilket kan være katastrofalt. En årsag til hvorfor forældrene ikke tager emnet om brandsikkerhed op med deres børn, er at de ikke selv har den nødvendige viden[4]. Hvis flere forældre selv vidste hvordan man skulle håndtere en brandsituation, ville de med større sandsynlighed videreføre denne viden.

Ifølge undersøgelsen er det kun 31 procent af danskere der har anskaffet sig ordentligt brandbekæmpelsesudstyr i form af brandslukkere og -tæpper osv. Dette vil selvfølgelig gøre det noget sværere at bekæmpe en allerede udbrudt brand.

De abiotiske faktorer også har noget at sige. Klimaforandringer i form af global opvarmning vil gøre naturen mere tør og tilbøjelig til at antænde[5]. Klimaforandringens påvirkning af naturen har især vist sig i denne sommer, hvor solen har været fremme mere end nogensinde før. Mellem den 1. maj og 10. juli rykkede det danske beredskab ud 1021 brandsituationer[6]. Dette er en stigning på 150 procent siden sidste år. Disse brande vil selvfølgelig være med til at øge faren for at komme til skade, men de vil endda også forårsage at tørkeperioder som disse forekommer hyppigere[7]. Naturbrande udleder nemlig CO_2 i atmosfæren, hvilket har været årsagen til at Klimaforandringerne startede. Dog er mængden af det udledte CO_2 meget lille, så påvirkning af klimaet er begrænset.

[1] (Beredskabsstyrelsen, 2018)
[2] (Beredskabsstyrelsen, 2018)
[3] (IDA, 2017)
[4] (Danske Beredskaber, 2017)
[5] (Sander, 2018)
[6] (Sander, 2018)
[7] (Jensen, 2018)

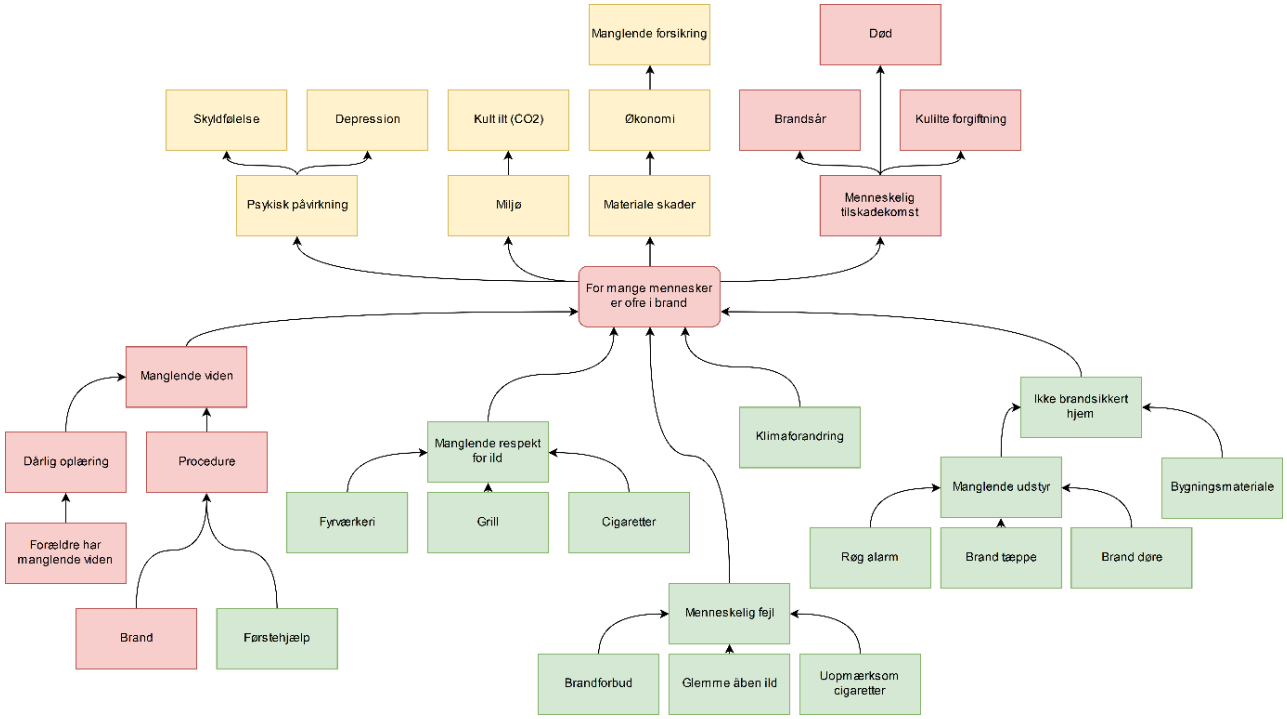
Interesseparter

Den almene person vil have en vis mængde interesse inden for dette emne. Specielt familier med børn har en ekstra interesse i at deres børn ved hvordan man agerer i brand, da det er dem problemet kan gå ud over. Derudover vil skoler såvel som andre undervisningsinstitutioner have interesse for at deres unge mennesker ved hvordan man skal forholde sig i den slags situationer.

Beredskaberne har interesse i at en så stor del af befolkningen som muligt kender til hvordan man skal forholde sig i tilfælde af brand eller ulykke. Deres interesse ligger på i effektiviteten af udførslen af deres arbejde, men også når de selv underviser på skoler og institutioner. F.eks. er de fleste beredskaber med i Beredskabsstyrelsens uge 40-kampagne, der handler om at sætte fokus på brandsikkerhed i de almene hjem[8].

Problemets sammenhæng og kompleksitet

På figuren herunder ses problemtræet for projektets overordnede problem. Dette giver et overblik over hvilke konsekvenser problemet har, og hvad der har ført til problemets opståen. Hovedproblemet befinder sig i midten af problemtræet. Kasserne over hovedproblemet repræsenterer problemets konsekvenser, og kasserne under repræsenterer problemets årsager. I dette projekt fokuseres der ikke på alle årsager ok konsekvenser, men udelukkende på dem markeret med rødt.



Figur 1 - Problemtræet. De rød-farvede kasser er dem vi har valgt at fokusere på i dette projekt

Projektrammerne

I forbindelse med undervisningen i Objektorienteret Programmering, skulle dette projekt udarbejdes. Der skulle igennem forløbet udvikles og dokumenteres et program ved hjælp af objektorienteret programmering. Programmet skulle bygges ud fra den udleverede kildekode til World of Zuul.
World of Zuul er et tekstbaseret spil, hvor brugeren interagerer med en konsolbaseret brugergrænseflade. Igennem forskellige kommandoer kan brugeren bevæge sig mellem forskellige rum.

Igennem projektet skulle World of Zuul-kildekoden udvikles, så der blev skabt et program, som passede til vores specifikke problemstilling, men stadig fulgte World of Zuul-konceptet med bevægelse igennem rum. Det

[8] (Beredskabsstyrelsen, 2017)

var blandt andre krav, at videreudviklingen skulle have flere rum, flere kommandoer, tekstbaseret brugergrænseflade (1. delforløb), grafisk brugergrænseflade (2. delforløb) osv. Disse rammer for projektet tilføjede både nogle forskellige begrænsninger og muligheder. I første delforløb, hvor programmet skulle være baseret på en tekstbaseret brugergrænseflade, blev programmet begrænset i dets evne til at beskrive sceneriet for brugeren. Dette blev nemmere i 2. delforløb med den grafiske brugergrænseflade. Derudover var projektet begrænset til at anvende det objektorienteret programmeringsparadigme. Objektorienteret programmering tilføjede dog på samme tid en stor mængde muligheder. At programmet skulle være objektorienteret, gjorde det muligt at lave klasser og objekter som gik igen igennem hele programmet. Man kunne altså genbruge store dele af koden og på den måde gøre programmet nemmere både at udvikle og vedligeholde.

8.2 Problemformulering

Brande har vist sig at være et stort problem i Danmark med ca. 70 branddrab og endnu flere kroner i skade om året. Der er flere årsager til problemets eksistens. F.eks. klimaforandringernes indtørring af naturen. Mere relevant er den manglende viden hos befolkningen. Især nutidens børn bliver ikke tilstrækkeligt udlært i hvordan de skal handle i tilfældet af brand. Dette er til dels forældrenes skyld, som heller ikke besidder den tilstrækkelige viden om brandsikkerhed. Dette problem er til gene for en stor del af befolkningen som f.eks. børnefamilier der er i fare for at problemet gør dem ude af stand til at handle tilstrækkeligt, hvis de selv skulle befinde dem i en brand.

I projektet skulle det simple tekstbaserede program World of Zuul videreudvikles til et stykke software som skulle kunne bidrage til løsningen af vores grundlæggende problem. Gruppen ønskede dermed igennem dette projekt at finde ud af om det kunne lade sig gøre, og hvordan softwaren i så fald skulle løse problemet.

Gruppens primære fokus på problemet var den manglende viden om brandsikkerhed blandt Danmarks indbyggere - både voksne og børn. Gruppen ønskede at skabe et program som ved hjælp af en kombination af underholdning og læring var i stand til at undervise børn og unge i hvordan man skal opføre sig hvis det brænder. Hovedspørgsmålet, som gruppen ønskede at besvare igennem dette projekt, er altså:

> Kan vi udvikle et læringsspil til børn og unge som kan bidrage til deres forståelse for hvordan brandsituationer skal håndteres, og som samtidig opfordrer til at der tages en snak om emnet i hjemmet?

For at kunne besvare projektets hovedspørgsmål, blev følgende spørgsmål nødt til at blive besvaret igennem projektet:

- Kan vi bygge et underholdende og læringsrigt program ud fra en World of Zuul-skabelon?
- Er det muligt at bygge et program som udspiller et scenarie hvor man befinder sig i en brændende bygning og skal finde ud?
- Kan man i programmet bygge en brændende bygning, så den implementerer rum-opdelingen fra World of Zuul?
- Kan man lave et børnevenligt spil i forhold til et meget seriøst og skræmmende emne som brandsikkerhed?
- Kan en form for konkurrence-element i form af point og/eller andet øge interessen for programmet?
- Skal programmet udlodde en gevinst når man har gennemført spillet for at øge motivationen og opfordre til en dialog i hjemmet?
- Kan man igennem spillet motivere en samtale om brandsikkerhedsemnet i hjemmet gennem direkte beskeder til brugeren?

Opgaverne der skulle løses igennem projektet for at kunne besvare hovedspørgsmålet:

- Få dannet os en forståelse for kildekoden.
- Få dannet nogle specifikke krav til programmet - både funktionelle og ikke-funktionelle.
- Få dannet et overblik over programmets designbindinger og -valg.
- Implementere flere funktionaliteter såsom forhindringer, genstande der kan samles op og derefter anvendes, personer/ting der kan interagere med brugeren (NPC).
- Lave løbende tests på koden.

8.3 Metoder

Dette projekt bygger på et praktisk problem, da løsningsmetoden er en konstruktion. Problemet har dog nogle teoretiske aspekter, da man bliver nødt til at foretage analyse og videns indsamling for at få en forståelse for problemet.

Når der skabes et program, gennemfører man fire forskellige grundfaser; krav, design, implementering og test. Disse er alle ting man bør overveje hvis man skaber en konstruktion. Krav-fasen består af indsamling og dokumentation af projektets funktionelle og ikke-funktionelle krav. Det er dem der sætter målene med programmet. Bliver disse overholdt har man et velfungerende program.

Herefter begynder designfasen. I denne fase bestemmes løsningsforslag til kravene. Det fastsluttes hvordan produktet skal udformes visuelt, arkitektonisk og teknisk. Man skelner mellem designbindinger og -valg, som henholdsvis er noget som kunden og programmøren beslutter.

Derefter kommer selve implementeringen af kode. Når man koder sit program, bliver man nødt til at overveje mange ting. F.eks. hvordan man vil navngive klasser, skrive kommentarer, formatere koden og mange andre ting.

Til sidst er det vigtigt at teste sit program. Det testes blandt andet om alle vores krav er overholdt, om kvaliteten af spillet lever op til både kundens og brugerens forventninger, og om ens kode overhovedet fungerer.

I den teoretiske del brugte gruppen søgningsværktøjer til at finde emnerelevante sider der kunne hjælpe med at opnå bedre indsigt i omfanget af problemet. Derudover tog gruppen også kontakt til brandvæsnet, som beskæftiger sig med forebyggende arbejder i forhold til brandsikkerhed. Dette gav et billede af deres syn på hvor stort omfanget af problemet er, og hvordan de tænker produktet kan være med til at afhjælpe problemet.

Moscow metoden

Der er gjort brug af Moscow metoden til at prioritere de forskellige krav der er sat til programmet. Det gør at man får et overblik over hvilke krav der er et Must-, should-, could- eller will not- have krav.

Must(M) have krav er alle de essentielle krav som skal til for at programmet giver mening.
Should(S) have krav er alle de krav som programmet burde have for at det er mest optimalt.
Could(C) have krav er alle de krav man kunne implementere hvis der opstod mere tid eller flere penge til at kunne implementere i programmet.
Will(W) not have krav som man vil fokusere på i fremtiden som ofte er urealistiske krav i den pågældende situation.

Gruppen brugte Moscow metoden for at kunne prioritere alle kravspecifikationerne, som gjorde det nemmere at planlægge arbejdsprocessen af arbejdsopgaverne i gruppen.

Kanban-board

Et kanban-board er en liste der er brugt til at tilføje Kanban metoden. Formålet ved brugen af Kanban metoden er til at kunne organisere opgaver under en arbejdsproces. Organisering af opgaver er ikke det eneste Kanban

hjælper med under arbejdsprocessen, men det hjælper også med at strukturere ens opgaver og have et overblik over ens arbejdsproces. Kanban er en metode der bruges sammen med en Kanban-board for at visualisere opgaverne.

Kanban-board bliver specielt brugt under softwareudviklinger, da det som tidligere nævnt hjælper med at visualisere arbejdsprocessen. Normalt bliver Kanban-board fordelt i tre kategorier; ”To Do”, ”Doing” og ”Done”. De tre kategorier er opgaver der skal færdiggøres, opgaver man er i gang med, og opgaver der er færdiggjort.

To Do	Doing	Done

Foroven ses eksemplet af en Kanban-board med de tre fordelte kategorier, hvor man opstiller opgaverne fra venstre og hen til højre. Kanban-board behøves ikke kun at bestå af de tre nævnte kategorier, men kan indeholde flere kategorier som ”Testing” for at give et overblik for ens softwareudviklingshold at man er i gang med at teste.

Parprogrammering

Når man programmerer, kan det ofte være kompliceret og svært at sidde alene om noget selv. Derfor findes der parprogrammeringsmetoden som kan bruges til at hjælpe andre med besvær for det bestemte programmeringssprog om at have en bedre forståelse for programmeringssproget, hvis man programmerede i par. Man kan ofte have forståelse for noget i programmeringssproget, men ikke alt, og det samme kan være for ens programmeringspartner. Parprogrammering er derfor en god programmeringsmetode for at dele ens viden og hjælpe hinanden med at få en bedre forståelse for programmeringssproget.

På grund af gruppens smalle kompetencer inden for programmering, var det udfordrende, hvis man skulle sidde alene om at udvikle funktionaliteter. Derfor brugte gruppen parprogrammeringsmetoden for at kunne hjælpe hinanden med at have en bedre forståelse af programmeringssproget.

Github

Github er en platform for softwareudviklere til at dele deres projekter. Disse projekter består af alle former for programmeringssprog. Projekterne kan privatiseres eller publiceres, så andre kan bruge koden. For at dele sit projekt, så skal der oprettes en repository, hvor alt ens indhold kan være derinde. Inde i sin repository står ens kode for sig selv i sit eget punkt, hvor man den ligger i hvad man kalder for ”master-branch”. Inde i ”master-branch” er den originale funktionelle kode. Det er derfor muligt at oprette andre branch for at viderearbejde koden i projektet uden at beskadige den originale kode. Når viderearbejdet er færdiggjort, kan man kombinere de to branch sammen til en helhed.

Andre softwareudvikler kan være med til at ændre på koden, men det kommer an på om ens repository er privatiseret eller publiceret. Hvis ens repository er publiceres, kan alle gå ind og viderearbejde koden, men skal ansøge om at kombinere deres ændring af koden med ”master-branch”. Men hvis koden er privatiseret, så skal man dele sin repository med andre udvikler man vil have med i projektet.

Github repository indeholder punktet insights, hvor man kan se fordeling på arbejdsbyrden. Dette er med til for at være sikker på, at hvis man arbejder i en gruppe, at alle har noget at tilbyde til projektet.

Da insights har været et godt overbliksværktøj af arbejdsbyrden, har gruppen brugt insights til at se hvem der har været med til at arbejde på koden i projektet, og om opgaverne er fordelt ligeligt mellem alle i gruppen.

Dette fik gruppen til at for eksempel begynde at parprogrammere for at kunne hjælpe alle med at have en forståelse for programmeringssproget og fordele opgaverne ligeligt.

8.4 Tidsplan

	Opstart					Fri	1. Iteration				2. Iteration					Fri
opgave/uge	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
samarbejdaftale	x	x														
vejlederaftale	x	x														
Idégenerering		x	x			E	x				x					J
problemanalyse		x	x	x		F										U
problemformulering			x	x		T										L
postersession					x	E										E
Projekt grundlag			x	x	x	R										F
gennemgang af World Of Zuul					x	Å	x									E
Kravspecifikation						R	x				x					R
konceptudvikling						S	x				x					I
Programmering						F		x	x	x	x	x	x	x		E
uml						E		x	x		x	x				
midtvejsseminar						R				x						
Rapportudkast						I			x	x						
processforbedringer						E					x					
Rapportskrivning												x	x	x	x	
Aflevering af spillet														x		

Figur 2: Overordnet tidsplan af projektforløbet

I starten af projektet lavede gruppen en tidsplan over hvordan tiden skulle fordeles i løbet af hele projektforløbet. Det blev delt op i 3 dele som hed opstart, første iteration og anden iteration. Alle tidsfrister for de forskellige afleveringer var offentliggjorte. Gruppen skulle nu gå igennem alle dele af projektet, for at forventning afstemme hvor lang tid der regnes med at skulle bruge på hver del.

9. Hovedtekst

9.1 Teori - Gamification

Gruppen undersøgte emnet gamification, og fandt frem til at det var når man benyttede sig af spilelementer, hvor det umiddelbart ikke hørte hjemme. F.eks. ville mange moderne læringsapplikationer indeholde point og highscores. Grunden til at gamification blev anvend så bredt var, at spilelementer blev designet til at kunne frembringe den menneskelige lyst til at socialisere, lære, konkurrere, opnå noget og meget mere.

Formålet ved brugen af gamification i dette projekt var, at det kunne vække interessen blandt brugerne. Mennesker havde tendenser til at være konkurrerende i spilbaseret emner. Derfor kunne det være interessevækkende og motiverende at tilføje spilelementer til læringsspil såsom et brandsikkerhedsspil, hvor målgruppen blandt andet er unge der kan var meget interesseret i et læringsmiddel der var baseret på gamification[9].

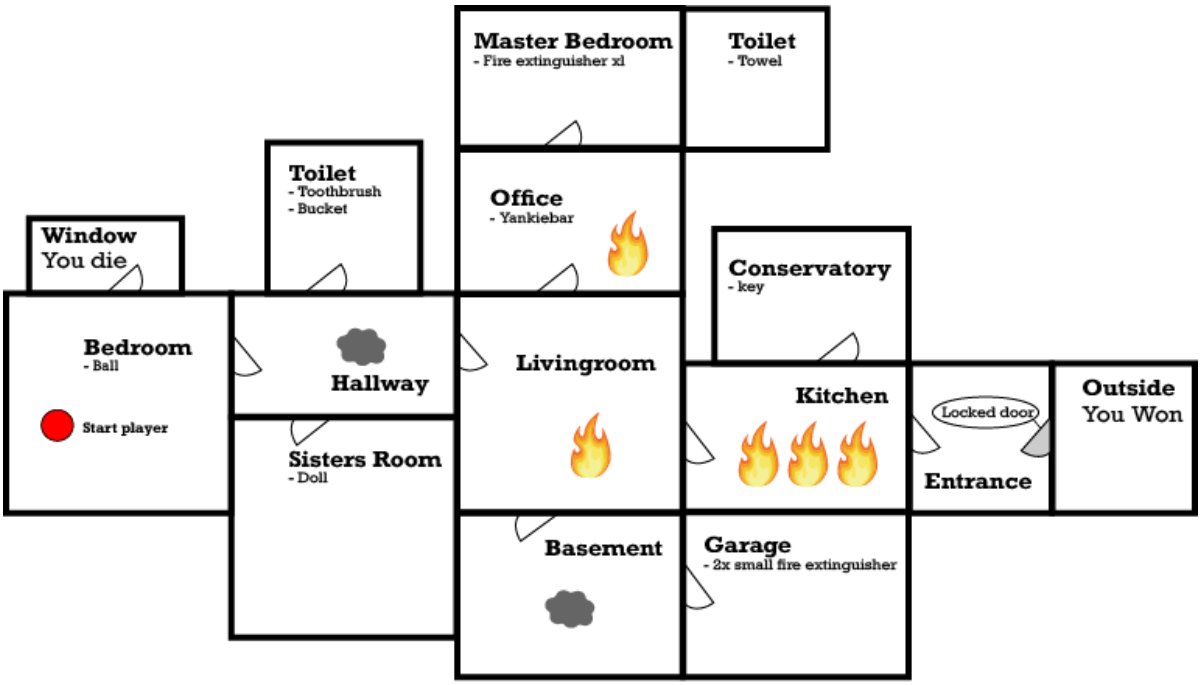
9.2 Spillets koncept

Spillets hovedkoncept gik ud på, at man skulle slippe ud af en brændende bygning. Denne bygning bestod af flere rum som spilleren skulle bevæge sig igennem før man kom til udgangen. I disse rum kunne der være forskellige forhindringer såsom ild, røg, og låste døre. Det blev så op til spilleren at finde veje udenom eller at fjerne forhindringerne. Dette kunne gøres ved hjælp af de værktøjer og objekter som spilleren havde til rådighed inde i de forskellige rum. Man havde vundet spillet hvis man fandt vej ud af bygningen, dog hvis man brugte for lang tid på det, ville man miste sit liv og tabe.

[9] (Ebdrup, 2015)

Tiden i spillet blev ikke baseret på tid som man kender den, men defineret for hver gang man skiftede rum, alle skridt tælles. Dette gjorde at man ikke skulle have travlt i hvert rum, så man kunne tage sig tid til at undersøge ens muligheder.

I starten fik man forklaret at der var ild i huset, man kender ikke vejen ud og man vidste ikke hvordan huset hængte sammen før at man så hvilke rum der var af muligheder at gå til fra det nuværende rum. Figur 3 herunder viser hvordan spillets rum er sat sammen og de objekter som de indeholder.



Figur 3: Et kort over vores spil

Player

Spilleren skulle kunne samle genstande op som kunne hjælpe spilleren ud, man kunne holde en ting ad gangen i sit **inventory**. Spilleren startere med 100 healthpoint. Hvis man igennem spillet mistede alle 100 ved f.eks. at komme i kontakt med ild, ville man tabe spillet og være nødsaget til at starte forfra. Spilleren havde muligheden til at bruge "help" til at få en forklaring af elementer som de ikke forstår.

Intro

Introduktion i spillet startede med, at spilleren vågnede op på sit værelse og kunne lugte røg. Der forklares at barnet tog telefonen og ringer til 1-1-2 og fortalte dem situationen.

De spurgte efter dit navn og det navn du indtastede, ville blive gemt til at lave en reference til din karakter i spillet. Man ville i resten af spillet, læse det som om det var en samtale mellem dit indtastede navn og 1-1-2. Hvis man brugte **help,** ville det være 1-1-2 som gav dig hjælp, for at gøre det mere livagtigt.

Item

For at spilleren kunne interagere med de forskellige items blev der lavet handlinger ved navn **search, take, drop** og **use.**

Handlingerne var de samme i begge iterationer ud over den. 1.iteration var tekstbaseret derfor skulle der forklares hvilke items der var i hvert rum og der skulle laves en ekstra handling der hed "search". Da 2.iteration var grafisk har denne handling ikke været nødvendig.

I 1.iteration skulle handlingen skrives manuelt efterfuldt af navnet på den item man vil interagere med. 2.iteration blev klikbaseret og det var derfor ikke nødvendigt.

Man kunne bruge "drop" som smed det objekt som var i dit inventar i det pågældende rum du var i, og det ville forblive i det rum med de samme værdier.

Når man havde samlet et objekt op, havde man mulighed for at bruge “inspect” til at få en beskrivelse af objektet og hvordan det bruges.
Spilleren skal komme forbi forskellige objekter som man finder i et normalt hus, der er implementeret objekter såsom tre **fireExtinguisher**, **bucket** og andre objekter samt ikke er brugbare objekter.

bucket kan bruges med **(use)** på toiletterne for at fylde spanden op med vand, hvis man har en spand fyldt med vand og bruger den i et rum der er ild i, vil ilden slukkes. Der er også objekter i spillet som kan gå ind og påvirke andre elementer som ens liv, der er placeret en **YankieBar** som vil gå ind og give dig fuld liv, så du kan holde dig i live inden du kommer ud af huset. **Yankiebarens** funktion er en af de elementer fra Gamification.

Forhindringer

Spillet indeholder forskellige forhindringer i løbet af spillet. Der er lavet 3 forskellige former for forhindringer, ild, røg og låste døre. De er alle placeret manuelt i de forskellige **Room** og kan alle flyttes rundt som man vil.

For at slukke ilden bliver man tvunget til at gå ind i det brændende rum og få påført sig noget skade for at kunne bruge vores objekter. Det er ikke muligt at gennemføre spillet uden at miste liv og det vil også betyde at man kan dø flere gange for at få et kendskab til spillets historie. Dette gør spillet udfordrende og fortæller også lidt om hvilke konsekvenser et brand kan have.

Der er forskellige niveauer på **Fire**, som er sat imellem 1-3, så man ikke komme igennem spillet uden at gøre brug af forskellige objekter. Det tvinger brugeren til at udforske spillet og tænke løsningsorienteret for at komme forbi forhindringerne. Der er sat en grænse på vores Fire niveau som er 3, da det skal være muligt at kunne gennemføre spillet i en passende sværhedsgrad.

Ud over den faste **Damage** på **Fire** er der yderligere også tilføjet en **UpdateFire** som øger ildens niveau for hver 5 step. Dette resulterer i at **Damage** stiger med 25 **hp** for hvert niveau man stiger. Det er gjort så det ikke er muligt at gå rundt frem og tilbage uden en konsekvens. **Damage** kan dog ikke komme over 75, i den samlede værdi sammen med **UpdateFire**. Niveau 3 Fire instansen kan dog ikke slukkes af **Bucket** eller af **FireExtinguisher** hvis det er den lille, da man skal bruge **FireExtinguisherXL** til at slukke **fire** på dette niveau.

Røg objekterne minder om **Fire**. **Smoke** er blevet lavet for at øge sværhedsgraden og for at få en bedre visualisering af situationen. I rum der indeholder **smoke** vil der tegnes billeder af røgen og spilleren mister 5 hp per niveau som røgen har. Alt **Smoke** har et fast niveau på 1, derfor skader det aldrig mere end 5 hp. Skaden er stadig baseret på det niveau som **Smoke** har, så sværhedsgraden i spillet nemmere kan justeres i fremtiden. I modsætning til **Fire**, kan røg ikke slukkes, fjernes på nogen måde og den kan heller ikke stige forhold til hvor mange steps man tager.

Den sidste form for forhindring er låste rum, alle Spillets rum har mulighed for at være låst. Spilleren vil ikke være i stand til at bevæge sig ind i låste rum. Spillet skal indeholde et eller flere objekter der kan åbne disse låste rum. Det er et lille tvist i spillet som ikke er brandrelateret, men som fortæller nogen af de situationer fra hverdagen der kunne blive et problem for spillerens overlevelse.

Highscore

I spillet laves en samlet highscore over hvert spil der spilles. Det betyder for hver gang du spiller et spil, vil den gemme de point du får indtil du enten dør eller vinder spillet. Igennem spillet udregnes brugerens score baseret på de handlinger der tages. Et eksempel på sådan en handling kunne være slukningen af ild. Hvis det

yderligere lykkes brugerne at gennemføre spillet ved at slippe ud af huset, tildeles en mængde ekstra-point. Den specifikke mængde ekstra-point afhænger af hvor mange skridt det har taget at nå ud. Jo færre skridt, desto flere point, da det handler om at slippe ud af bygningen så hurtigt som muligt. Disse point gemmes i en liste i en separat tekstfil.

Highscore-implementeringen motiverer brugerne til at blive bedre og udfordre sig selv samt hinanden. I hovedmenuen inden spillet starter kan man trykke på highscore knappen og får printet vores highscore fil så den læsser de 10 bedste navne og får vist hvor mange point man skal opnå for at være bedre.

9.3 Krav

For at danne et grundlag for hvordan et program skal fungere, og hvad det skal kunne, kan man opstille krav. Kravene inddeles i to kategorier: funktionelle og ikke-funktionelle. Disse krav kan bruges til at konkretisere hvad programmet skal være i stand til og have af kvaliteter.

Funktionelle krav

De funktionelle krav illustrerer programmets/systemets funktionalitet. Til projektet er der defineret følgende funktionelle krav:

ID	Krav	Detaljer	Kilde
F1	Bevæge sig imellem rummende.	Brugeren skal kunne skifte imellem de forskellige rum i spillet.	Projektoplægget
F2	Brugeren skal kunne indtaste kommandoer som spillet reagerer på.	Brugeren skal kunne indtaste en række forudbestemte kommandoer for at spille spillet.	Projektoplægget
F3	Spillet skal kunne gennemføres.	Det skal være muligt for brugeren at komme ud af bygningen, med det ene mål betyder det så at spilleren kommer ud i sikkerhed.	Gruppen
F4	Spillet skal indeholde objekter som man kan interagere med og samle op.	De kan f.eks. være til for at hjælpe brugeren med at komme forbi forhindringer. Yderligere skal der være redskaber der bruges til brandsikkerhed så brugerne på samme tid blive undervist i hvordan de virker.	Projektoplægget
F5	Forhindringer der kan fjernes ved brug af objekter	At spillet indeholder forhindringer, gør det mere spændende og udfordrende. Dette skulle gerne øge interessen for det hos målgruppen.	Gruppen
F6	Spillet skal kunne tabes.	At gøre brugeren i stand til at tabe spillet med f.eks. tid/liv. Vil være med til at gøre spillet mere udfordrende.	Gruppen
F7	Spillet skal indeholde et point-system.	At indføre et point-system som afspejler hvor godt brugeren har klaret sig, til spillet vil tilføje et konkurrenceelement.	Gruppen
F8	Spillet skal indeholde non-player character(s).	Spilleren skal kunne interagere med en NPC. De kan hjælpe med at give brugeren flere muligheder og generelt bare udvide spillets indhold. I vores projekt kunne disse eventuelt bruges til at formidle viden om brandsikkerhed til brugeren af spillet.	Gruppen
F9	Spillet skal have en grafisk brugergrænseflade.	I spillet skal være GUI. Det betyder at der skal laves visuelle elementer til alle objekter i spillet.	Projektoplægget
F10	Brugeren skal kunne inteagere og navigere rundt i spillet ved hjælp af grafiske elementer	Spilleren skal udelukkende ved hjælp af de grafiske elementer kunne gennemføre spillet.	Gruppe

Ikke-funktionelle krav

De ikke-funktionelle krav beskriver hvilke kvaliteter et program skal besidde. Til projektet er der defineret følgende ikke-funktionelle krav.

ID	Krav	Detaljer	Kilde
B1	Spillet skal være brugervenligt for børn fra 8 år og op.	At programmet er brugervenligt, vil selvfølgeligt gøre det nemmere at finde ud af og anvende. Brugerne vil gerne have et program der er nemt at bruge.	Gruppe
B2	Spillet skal have en tilstrækkelig ydeevne.	At programmet ikke opfører sig langsomt eller går i hak, vil være med til at øge dets kvalitet.	Gruppe
B3	Spillet skal være nemt at modificere (godt design).	Hvis et program ikke er nemt at modificere, kan det både tage længere tid og ende med flere fejl, hvis man senere hen i eller efter projektet beslutter sig for at redigere/opdatere det.	Gruppe
B4	Brugeren skal udfordres i sin problemløsningsevne.	Spillets brugere skal tænke og løse problemer. Dette vil gøre det mere spændende og udfordrende.	Gruppe
B5	Brugeren skal gennem spillet undervises i og bruge sin viden om brandsikkerhed.	At brugeren faktisk får noget viden og læring med sig fra vores program, vil være med til at løse det overordnede problem, som vores program egentlig forsøger at løse.	Gruppe

ID	Iteration	Prioritet
F1	1 & 2	M
F2	1	M
F3	1 & 2	S
F4	1 & 2	M
F5	1 & 2	S
F6	1 & 2	S
F7	1 & 2	S
F8	1 & 2	C
F9	2	M
F10	2	M

ID	Iteration	Prioritet
B1	1 & 2	S
B2	1 & 2	S
B3	1 & 2	S
B4	1 & 2	S
B5	1 & 2	S

Delkonklusion

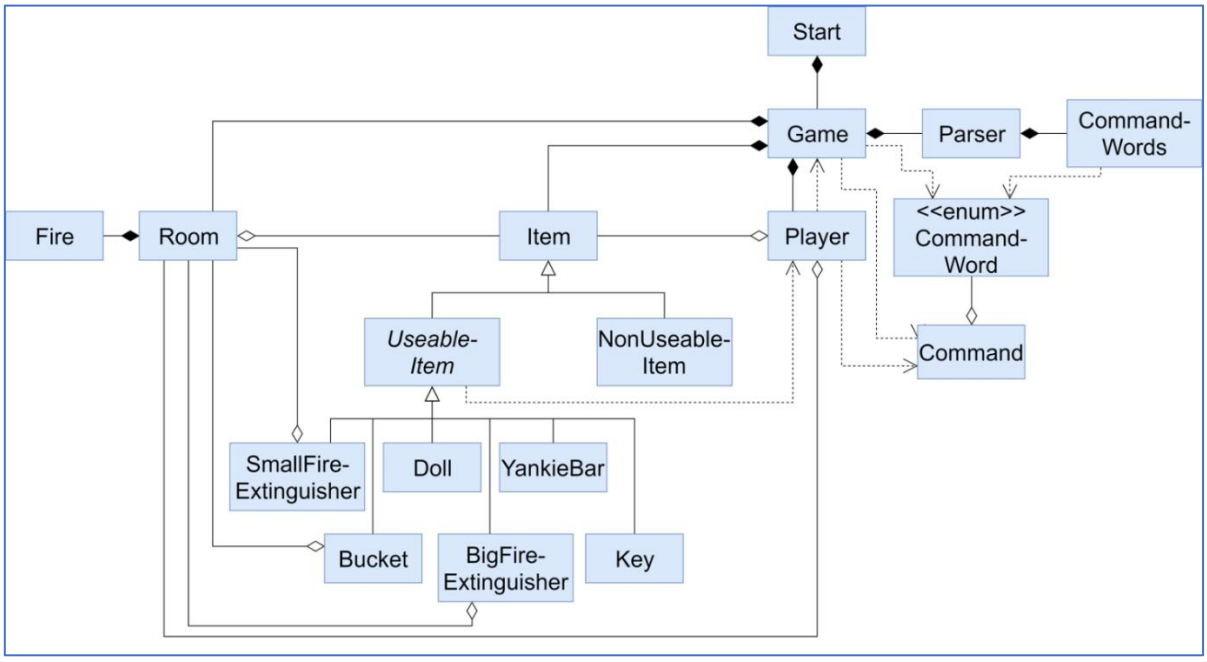
Da kravspecifikationen var fastlagt, havde gruppen en idé om hvad programmet skulle bestå af. Disse krav sammen med deres prioritering kunne hjælpe gruppen med at udvikle programmet i en rækkefølge, så det vigtigste blev håndteret først.

9.4 Design

Hvordan et programs enheder og komponenter er sat sammen, kan sige meget om dets kvalitet. Det er derfor vigtigt at have fokus på et systems design, både overordnede og detaljerede, før såvel som efter systemet opbygges. I dette afsnit kigges der på designet af dette projekts produkt. Designvalgene for både første og anden Iteration undersøges og forklares.

Første iteration

Figur 4 herunder viser det overordnede design af programmet efter første iteration. Hver kasse på figuren repræsenterer en klasse i programmet. Forbindelserne imellem illustrerer hvordan klasserne interagerer med hinanden. Figuren er et simplificeret klassediagram som undlader attributter, metoder og multiplicitet.



Figur 4: Klassediagram af Fire Escape:

Game

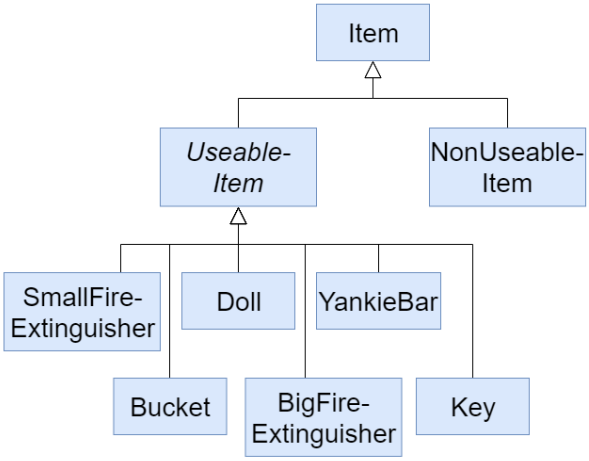
I første iteration var **Game**-klassen et samlingspunkt for resten af klasserne. **Game** var klassen der skabte **Player**, alle **Items**, alle **Rooms** og **Parseren**.

Player

Player-klasse var ikke med i den oprindelige World of Zuul-kode. Den blev oprettet for at samle funktionalitet. Funktionalitet som **goRoom**-, **currentRoom**- og **nextRoom**-metoden som før var i **Game**-klassen hørte efter første iteration til **Player.** Dette gav bedre mening, da det var spilleren der befandt sig i og bevægede sig mellem de forskellige rum. Denne sammenhæng mellem klassen **Player** og klassen **Room** er på klassediagrammet illustreret med en rombe-formet pil uden fyld. Denne pil indikerer *Aggregering*, hvilket betyder, at en instans af **Room** er en del af **Player.** Derudover havde **Player**-klassen også en attribut af typen **Item** ved navn **inventory**. Denne opbevarede det objekt som spilleren havde samlet op. Dette er endnu et eksempel på funktionalitet som hørte mere til **Player**-klassen. På klassediagrammet er forbindelsen imellem **Player** og **Item** igen indikeret med en rombeformet pil der ikke er udfyldt.

Item

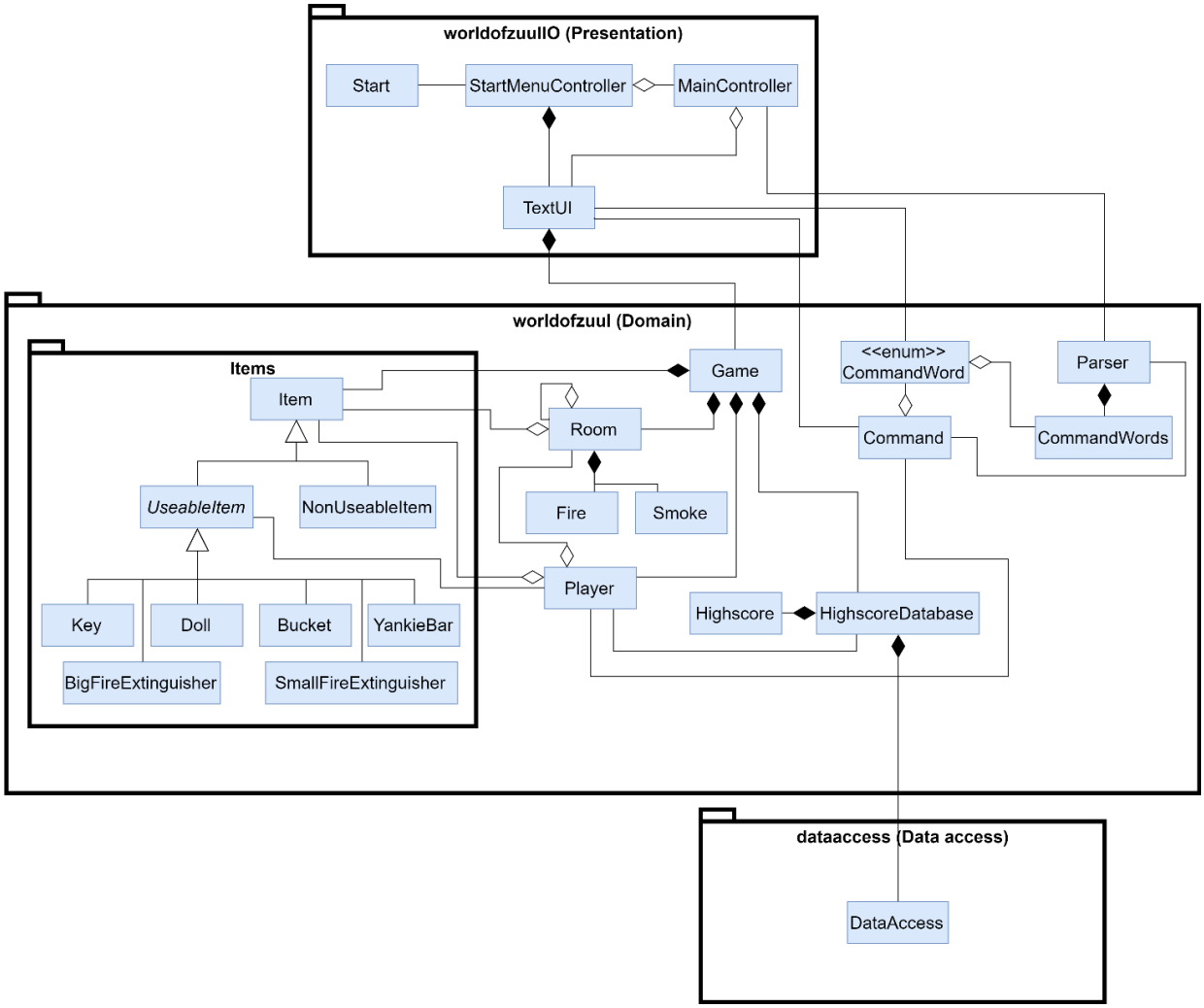
Figur 5 til højre viser en isoleret version af hele Item-klasse-strukturen. På denne figur ses, at klassen **Item** havde to underklasser; **UseableItem** og **NonUseableItem**. Spilleren skulle være i stand til at samle forskellige typer af objekter op og anvende disse. For at sikre, at hver eneste **UseableItem** fik sine egne unikke funktionalitet, blev der lavet en klasse for hver type **UseableItem**. På figur 3 ses disse klasser, som også nedarver fra klassen **UseableItem**. Indtil videre indeholder vores spil altså hele seks forskellige **UseableItems**; **Bucket**, **SmallFireExtinguisher**, **BigFireExtinguisher**, **YankieBar**, **Key** og **Doll**. For at sikre, at de alle implementerede deres egne funktioner for hvordan de anvendes, blev **UseableItem** gjort til en abstrakt klasse. Her blev den abstrakte metode **use(Player player)** tilføjet. At **UseableItem** var en abstrakt klasse gjorde, at man ikke var i stand til at instantiere den. At tilføje den abstrakte metode **use** betyder, at hver klasse der arver fra **UseableItem** blev nødt til at definere sin egen version af denne metode. Det betyder at alle **UseableItems** implementede forskellige funktionaliteter af **use**-metoden.



Figur 5: Lille udkast fra en simpel version af klassediagrammet

Anden Iteration

På figur 6 ses det overordnede design efter projektets anden iteration. Den mest tydelige forskel er, at sammenhængende klasser nu er samlet i pakker. Programmets overordnede arkitektur er nu lagdelt i tre lag; præsentation, domæne og data.



Figur 6: Lagdelt arkitektonisk UML-diagram

Præsentation

Øverst ses mappen **worldofzuulIO** som repræsenterer præsentationslaget. Indholdet af denne har udelukkende at gøre med håndtering af hvad der skal printes og tegnes på skærmen til brugeren, og hvordan brugerens input til programmet skal fortolkes. I denne mappe findes nye klasser. F.eks. ses der to Controllere. Disse bestemmer hvad programmet skal gøre i tilfældet af, at brugeren trykker på en knap eller lignende. Der er to controllere, og den første knytter sig til programmets startmenu. Her er der en mængde knapper som brugeren kan anvende til at navigere rundt i programmet med for f.eks. at starte det endelige spil. Hvis brugeren vælger dette, gives kontrollen videre til **MainController**, som står for at håndtere alle brugerinput når det egentlige spil starter.

Som diagrammet viser, skabes der en instans af **TextUI** inde i **StartMenuController**. **TextUI** er den klasse der står for at håndtere al udskrift af tekst til skærmen. Den bruges også til at fortolke de kommandoer der sendes fra Controllerne, når brugeren trykker på knapper. **TextUI** indeholder en instans af **Game**. **Game** er placeret i domænelaget, og har mere eller mindre samme rolle som før. Som man kan se på diagrammet, fungerer **Game**

stadig som et samlingspunkt for spillets mest centrale klasser. **Player**, **Room**, **Items** og **HighscoreDatabase** er alle enheder som skabes inde **Game**-klassen. At **TextUI** skaber **Game**, gør at **TextUI** fungerer som den primære forbindelsen mellem præsentationslaget og domænelaget.

Domænelag

Domænelaget er ikke ændret markant siden sidste iteration ud over tilføjelsen af et par ekstra klasser, en pakke til **Item** med subklasser og nogle lidt nye forbindelser.

Highscore

Highscore og **HighscoreDatabase** er et par af de nye klasser. Som fortalt, gemmer spillet brugerens highscore, når spillet er ovre. Dette gemmes i en tekstfil i projektmappen. Inde i **Game** oprettes en **HighscoreDatabase**. Denne har adgang til tekstfilen med highscores igennem **DataAccess**-klassen der er placeret i det nederste lag af programmet (dataaccess-pakken). **DataAccess** står for al skrivning og læsning til og fra filer. Igennem denne er **HighscoreDatabase** i stand til at gemme en mængde **Highscores** i en liste. Hvis brugeren afslutter et spil, vil **HighscoreDatabase** tilføje en ny **Highscore** og give besked til **DataAccess** om at gemme den nye liste i tekstfilen.

Item

Derudover er hele **Item**-klassen og alle dens subklasser flyttet over i en pakke for sig selv. Denne pakke er dog stadig en del af domænelaget. Alle former for **Items** er isoleret, da de har en logisk sammenhæng. Denne isolering er med til at gøre programmet mere overskueligt. Det samme kan siges om ændringen i programmets overordnede arkitektur. At de klasser der har lignende funktioner er samlet i pakker isoleret fra de andre, gør både programmet mere overskueligt og nemmere at redigere i senere hen.

Genbrug

En stor del af opbygningen fra 1. iteration er genbrugt i 2. iteration. F.eks. bruger programmet stadig en **Parser** til at afkode de kommandoer programmet får ind. I 1. iteration havde brugeren mulighed for at indtaste en vilkårlig mængde af valgfrie tegn ind i programmet, og her giver det mening at have en **Parser** som afkoder inputtet. I 2. iteration der anvender en grafisk brugergrænseflade, hvor brugeren interagerer ved at klikke på knapper, er der et begrænset antal muligheder for brugerinputtet. Det er nu ikke længere muligt for brugeren at skrive nogle kommandoer ind igennem tastaturet. **Parserens** nødvendighed kan derfor diskuteres. Dog gjorde det udviklingen af den grafiske brugergrænseflade noget nemmere. Efter første iteration kunne programmet nemlig tyde nogle særlige kommandoer fra brugeren og handle som følge heraf. Når brugeren trykker på en knap på den grafiske brugergrænseflade, lades der som om, at der indtastes en specifik kommando på samme måde som i første iteration, og den samme funktionalitet opretholdes. Dette var en nemmere måde at omdanne programmet til GUI, da funktionalitet der i forvejen virkede kunne genbruges.

Delkonklusion

Mellem første og anden udgave af programmet, skete der store ændringer i den overordnede arkitektur. Den anden iteration blev opbygget efter en lagdelt arkitektur med tre lag, hvilket gav nogle markante fordele. Det vil f.eks. blive nemmere at modificere programmet i fremtiden, da man nu kunne nøjes med at udskifte et lag i stedet for at skulle omskrive hele koden. Manglen på lagdeling kom især til udtryk, da anden iteration af programmet skulle udvikles. Da der ikke var nogen særskilt afdeling af programmet som stod for brugerinput og -output, var det mere indviklet at omskifte fra CLI til GUI. Dog gjorde den gode klasseinddeling og strukturering af 1. iteration arbejdet noget nemmere, da gruppen var i stand til at bygge den grafiske brugergrænseflade omkring det allerede eksisterende domænelag.

9.5 Implementering

Koden blev bygget op ud fra idéen om at den skulle være let læselig, og let at modificere, for at sikre at gruppen og andre, effektivt kunne tilføje funktionalitet og rette fejl.

I hele kodningsprocessen blev der blevet fulgt Javas Camel Case metode, som gjorde navne på for eksempel metoder nemmere læselig, ved at hvert nyt ord blev startet med stort bogstav for at man nemmere kunne skille hver enkelt ord fra hinanden i metodenavnet. Hvis man fulgte denne logik, gjorde det også de fleste metoder selvsigende, så man bare ved at læse metodenavnet kunne se hvad en pågældende metode gjorde, uden at behøve ekstra kommentarer i koden.

I koden er der sørget for at holde metoderne korte (max 50 linjer (dog med få undtagelser)) for at undgå fejl som kan være forbundet ved at lave lange komplicerede syntaks, længere syntaks kan være acceptabelt, hvis man i stedet slipper for at duplikere kode. I forhold til duplikering af kode, så har gruppen skrevet koden på den måde at den samme kode helst ikke skal stå flere steder i programmet, og derfor også prøvet at finde flere muligheder for at omskrive det duplikerede kode til en metode, som varetager samme funktion som den duplikerede kode ville, med det resultat at der skal spares linjer og den vej igennem have en forbedret kode, hvis koden senere hen skulle vedligeholdes eller på anden måde gennemgås.

Mellemrum og indryk

Gennem hele kodningsprocessen blev der overholdt de indryk som gav mening i forhold til koden. Dette var noget NetBeans IDE hjalp med, men det er dog vigtigt at man selv havde styr på indryk og mellem for at sikre læsbarheden af funktionerne. I forhold til indryk blev der sørget for at når man skiftede linje, så laves der indryk så det passede til det der blev implementeret, for eksempel blev koden også indrykket, som hvis man for eksempel lavede et if-statement inde i et for-loop, så blev denne rykket længere ind end selve for-loop var. For at forbedre læsbarheden af koden blev der også blevet brugt mellemrum. Mellemrummene blev placeret så at koden i Java tillader at man skriver en betingelse for et if-statement, sammensat uden mellemrum, så blev der lavet mellemrum, så betingelsen var nemmere at læse.

På figur 7 ses koden med mellemrum og indryk, og på figur 8 ses et eksempel på samme kode uden

```
public void updateFire(Game game) {
    if (getStepCount() % 5 == 0) {
        System.out.println("The fire got bigger...");
        for (Room room : game.getRooms()) {
            room.updateFire();
        }
    }

}
```

Figur 7: Kode med mellemrum og indryk

```
public void updateFire(Game game) {
if (getStepCount()%5==0) {
System.out.println("The fire got bigger...");
for (Room room:game.getRooms()) {
room.updateFire();
}
}
}
```

Figur 8: Kode uden mellemrum og indryk

Koden manglede kommentarer, derfor når 2. iterationen startede blev det vigtigt at gruppen valgte at få implementeret kommentarerne, så de passede til koden, sådan at kommentarer kun skrives i det omfang de var nødvendige, og ellers udelades for metoder som var selvsigende.

Spillet blev implementeret med mange forskellige funktioner, men for ikke at gøre det for fyldigt blev der udvalgt vigtige implementeringers eksempler fra spillekoden herunder:

Item

Herunder ses koderne for hvordan **Item**-klassen fungerer. **Item-**klassen blev brugt i **Game-**klassen til at oprette objekter brugeren kunne interagere med. Når objektet var blevet oprettet, så skal rummene blive tildelt et objekt, og det gøres på figur 10. Objekterne skulle være mulige at interagere med. Derfor blev der oprettet to sub-klasser, **UseableItem** og **NonUseableItem**. Disse to klasser separerer objekter der kunne bruges igennem spillet, og objekter der var tilstede for at give brugeren korrekte eller forkerte valgmuligheder.

```
private void createItems() {
    bucket = new Bucket("Bucket", "Holds liquid well.
    toothbrush = new NonUseableItems("Toothbrush", "M
    smallFireExtinguisher = new SmallFireExtinguisher
```

Figur 9: createItems()-metoden bruges til at oprette nye items

```
wc.addItem(bucket);
wc.addItem(toothbrush);
wc2.addItem(towel);
```

Figur 10: Rummene bliver tildelt items

```
@Override
public void use(Player player) {

    System.out.println("*OM NOM NOM*");
    System.out.println("Your health has been restored to 100.");
    player.removeItem();
    player.setHealth(100);
}
```

Figur 11: Use()-metoden som oprettes i programmets abstrakt klasse, UseableItem

Use()-metoden blev kaldt i brugerens inventar ved at kontrollere efter objekter i brugerens inventar.

Opsamlingen af objekter foregik i **Player**-klassen. Herinde oprettede gruppen metoden **takeItem(Command command)**. Denne blev kaldt fra **Game** hver gang brugeren skrev kommandoordet *take* i CLI-versionen. Denne metode tjekkede først om brugeren havde indtastet navnet på den ting de vil samle op. Hvis ikke, skrev programmet "*Take what?*" ud til brugeren og afbrød metoden med en **return**. Hvis spilleren havde tastet et navn, ville metoden så tjekke om der var plads i spillerens **inventory**. Hvis dette var tilfældet, ville metoden fortsætte med selve opsamlings-processen. Dette foregik igennem et for-loop som kan ses på figur 12. Inde i for-loops initialisering oprettes tællevariablen **i** som sættes lig med nul. Stopbetingelsen for dette for-loop sagde, at **i** skulle være under antallet af **Items** i det rum som spilleren stod i på det pågældende tidspunkt. For-loopet itererede igennem ArrayListen af **Items** som rummet **currentRoom** indeholdte. For hver iteration tjekkes det om navnet på **Item** stemte overens med hvad brugeren havde indtastet i programmet (**itemName**). Hvis dette var tilfældet, ville dette objekt blive gemt på variablen Inventory.

```
for (int i = 0; i < currentRoom.getItems().size(); i++) {
    if (itemName.toUpperCase().equals(
            currentRoom.getItems().get(i).getName().toUpperCase())) {
        inventory = currentRoom.getItems().get(i);
        System.out.println("You pick up the "
                + currentRoom.getItems().get(i).getName() + ".");
        currentRoom.getItems().remove(i);
        return;
    }
}
```

Figur 12: For-loop for indsamling af item

Når spilleren skrev *"drop"* inde i programmet, ville **Player** fjerne sin opsamlede **Item** fra **Inventory**. Det var ikke nødvendigt at skrive navnet på den item som skulle droppes, da **Player** kun var i stand til at bære på én ting af gangen.

Gruppen lavede i 2.iteration en funktionalitet der gjorde det muligt for brugeren at samle et objektet op og tilføje det til sin **Inventory** ved at klikke på den pågældende item. Dette krævede at gruppen kombinerede et billede med en **EventHandler**. Koden ses i figur: 13

```
private void printItems() {
    for (Item item : textUI.getGame().getPlayer().getCurrentRoom().getItems()) {
        ImageView img = item.getImage();
        paneRoom.getChildren().add(img);

        img.setOnMouseClicked(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent e) {
                if (textUI.getGame().getPlayer().getInventory() == null) {
                    imgInventory.setImage(img.getImage());
                    paneRoom.getChildren().remove(img);
                }
                processCommand("take " + item.getName());
            }
        });
    }
}
```

Figur 13: printItems()-metoden tegner Item i rummene

PrintItems()-metoden kontrollerede hvilke **Items** der var tilknyttet til det nuværende rum, og tilføjede derefter billeder af disse **Items** til rummet. Hvert enkelt billede havde derefter en **EventHandler** der, når der blev trykket på et **Item,** kontrollerede om spillerens **Inventory** var tomt. Hvis det var tilfældet, blev billedet af objektet tilføjet til spillerens visuelle **Inventory** og billedet blev fjernet fra rummet. "take" Kommandoen blev kaldt på det pågældende **Item**, for at objektet blev tilføjet til spillerens **Inventory.**

Fire

Ilden i spillet blev defineret ud fra **Fire**-klassen. **Fire**-klassen bestod af en constructor, hvor man indsatte en integer værdi som level der bestemte ildniveauet. I **Fire**-klassen fandtes en metode der opdaterede ild-niveauet konstant, men da ildniveauet kunne stige uendeligt, blev der tilføjet et if-statement der tvang **lvl**-attributten til maksimalt at nå 3.

```
public class Fire {

    private int lvl;

    public Fire(int lvl) {
        this.lvl = lvl;
    }

    public int getLvl() {
        return lvl;
    }

    public void updateLvl() {
        lvl++;
        if (lvl > 3) {
            lvl = 3;
        }
    }
}
```

Figur 14: Fire-klassen

```
public void addFire(int lvl) {
    fire = new Fire(lvl);
}

public Fire getFire() {
    return fire;
}

public void removeFire() {
    fire = null;
}

public void updateFire() {
    if (fire != null) {
        fire.updateLvl();
    }
}
```

Figur 15: Firerelaterede metoder i Room-klassen

```
public void updateFire(Game game) {
    if (getStepCount() % 5 == 0) {
        System.out.println("The fire got bigger...");
        for (Room room : game.getRooms()) {
            room.updateFire();
        }
    }
}
```

Figur 16: Metode der opdaterer ild i Player-klassen

```
private void createFire() {
    kitchen.addFire(3);
    office.addFire(1);
    livingRoom.addFire(1);
}
```

Figur 17: Tildeling af ild i bestemte rum

I **Room**-klassen blev der oprettet nogle **Fire**-metoder til at kunne tilføje ild i rummene som kan ses på figur 15. Her var de vigtigste metoder, blandt andre, **addFire()**-metoden og **updateFire()**-metoden. **addFire**-metoden blev brugt til at tilføje ild i de forskellige rum som kunne gøres i **Game**-klassen. **updateFire()**-metoden blev brugt til at opdatere ild niveauet i rummet. **addFire**-metoden blev brugt til at tilføje ild i et rum, og det blev gjort i **createFire()**-metoden i **Game**-klassen. Endnu en **updateFire()**-metode blev oprettet i **Player**-klassen for at kunne opdatere ild niveauet i forhold til antal skridt man har foretaget. Denne metode fortalte spilleren når ild niveauet steg, og har et for-loop der tjekker om der var ild i rummet og hvornår denne skulle opdateres til **updateFire()**-metoden i **Room-**klassen.

Fire-klassen blev kun oprettet til at tjekke for ild i det givne rum og ildens niveau.

```
public Player(Room room, String playerName) {
    stepCount = 0;
    health = 100;
    inventory = null;
    currentRoom = room;
    this.playerName = playerName;
}
```

Figur 18: Player constructor

```
public int takeDamage(int dmg) {
    return health -= dmg;
}
```

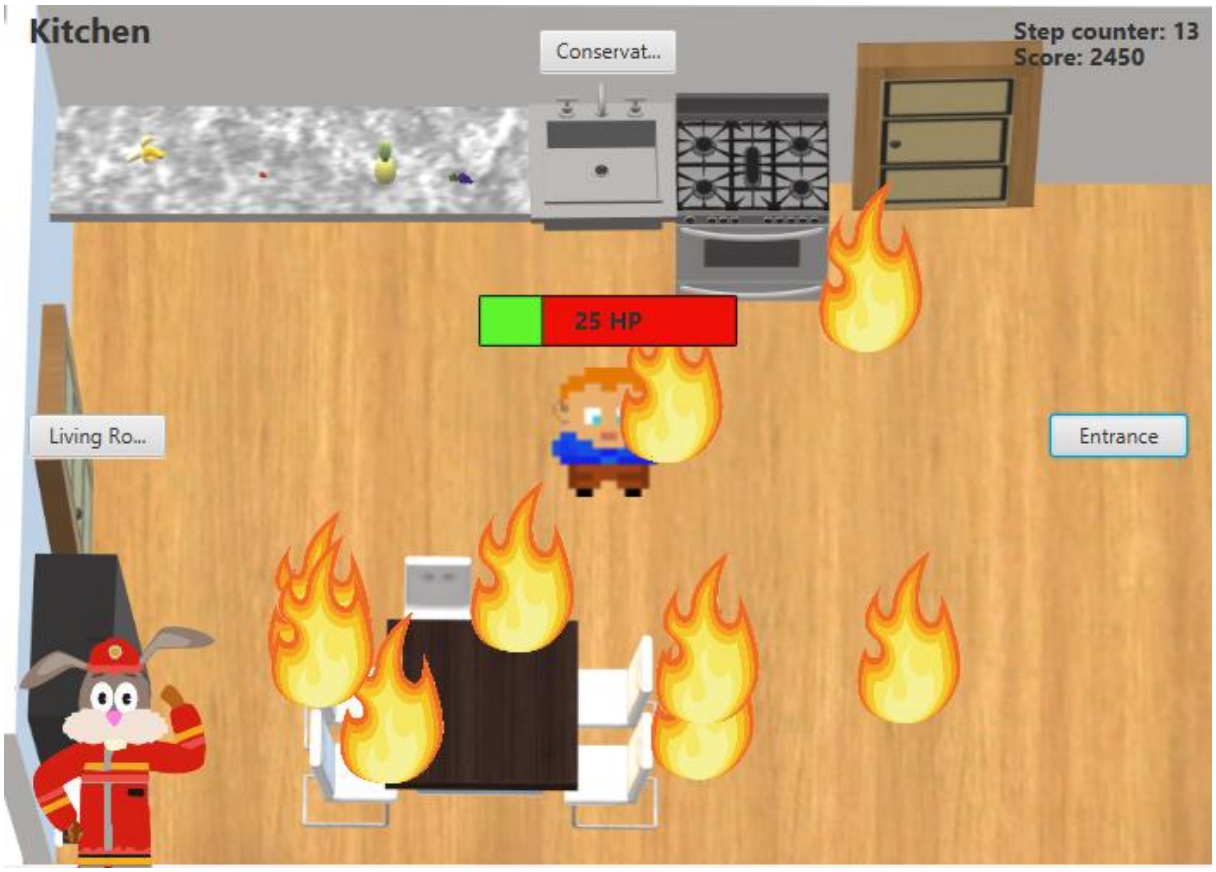
Figur 19: takeDamage-metode i Player-klasse

For at distribuere skade til spilleren, så blev der til at starte med tildelt attributten “health” til Player-constructor, som holdte styr på spillerens liv, for at gøre det muligt at tabe spillet. For at spilleren kunne tage skade, blev der oprettet en metode som blev kaldt **takeDamage(int dmg)**. Denne metode returnerede “health”

minus det indtastet "dmg". Spilleren ville tage 25 skade hvis der var ild i rummet, afhængig af ild niveauet, som blev udregnet ved at gange 25 med **currentRoom.getFire().getLvl()**. Ildniveauet kunne variere fra 1 til 3.

I 1. iteration blev der indført en base-damage til hvert rum. Dette var en ekstra skade der blev lagt til ild-skaden. Den blev indført for at sikre spillerens vej rundt i spillet. Den viste sig ikke at være nødvendig, men blev beholdt efter at have testet koden og ændret lidt på funktionerne, da denne i 2. iteration kunne være nødvendigt. Igennem 2. iteration blev den ikke nødvendig, så base-damage blev fjernet fra programmet.

For at visualisere ilden i den grafiske brugergrænseflade er der blevet tilføjet et billede af ild i de rum der indeholder ild. For hvert ildniveau vil der være tre ild billeder. Det kan for eksempel ses på figur 20, hvor spilleren står i køkkenet med et ildniveau på 3, og der vises 9 ild billeder.



Figur 20: Illustration af spilleren i et rum med brand

Billederne blev oprettet som et "Image" i **Fire-**klassen, men blev oprettet som en "ImageView" i en metode i **MainController**-klassen for at kunne visualisere billedet i spillet. Metoden i **MainController**-klassen, hvor billedet af ilden blev oprettet som en "ImageView" er blevet programmeret med JavaFX, og ikke FXML, da det var en mindre opgave, og simplere at lave ved brug af JavaFX.

```
private void printFire() {
    Fire fire = textUI.getGame().getPlayer().getCurrentRoom().getFire();
    if (fire != null) {

        for (int i = 0; i < fire.getLvl() * 3; i++) {
            ImageView imgFire = new ImageView(Fire.IMAGE_FIRE);
            imgFire.fitHeightProperty().set(100);
            imgFire.fitWidthProperty().set(60);
            imgFire.setTranslateX(80 + Math.random() * 480);
            imgFire.setTranslateY(30 + Math.random() * 330);
            paneRoom.getChildren().add(imgFire);
        }
    }
}
```

Figur 21: printFire()-metoden indsætter billeder af ild inde i rumme med ild.

Billedet af ilden blev tegnet i forhold til hvad ildniveauet i rummet er på. Man kan f.eks. se på figur 21, at metoden startede med at tjekke om der er ild i rummet, hvis der er ild i rummet, så ville for-loopet tjek hvad ild niveauet er ganget med 3.

For at få billedet af ilden indsat i rummene i spillet, så bliver **printFire()**-metoden indsat i metoden **redrawRoom()**, som kan ses på figur 24, side 30.

Player Movements

I command-Line interface versionen af brandsikkerhedsspillet kunne spilleren bevæge sig ved at indtaste nogle kommandoer som ”go north”, ”go south”, ”go east” og ”go west”. Disse kommandoer bestod af to sammensatte Strings der tilsammen ville fungere som et kommando. Den første String var ”go” og den anden String var hvilken retning, altså ”north”, ”south”, ”east” eller ”west”. Hvert rum blev tildelt et ”exitString”, som fungerede på den måde at der tildeles en retning til det andet rum. Eksemplet kan ses på figur 22**.**

```
bedroom.setExit("east", hallway);
bedroom.setExit("north", window);
```

Figur 22: Rummene bliver tildelt udgange

På figur 22 kan man se at ”bedroom” blev tildelt to udgange som er ”hallway” og ”window”, men når man havde udgange, så skulle man også have retningen, som også kan ses, er blevet tilføjet.

I graphic interface versionen var der dog ikke så stor forskel, da man stadig brugte disse kommandoer til at bevæge sig fra rum til rum. Men for at gøre det blev der blevet oprettet en metode der læste kommandoerne ”go” + retning. Denne metode hed processCommand(String inputLine), som fungede på den måde at man skrev kommandoen for en given retning som String. Dette kan for eksempel ses på figur 23, hvor ”north”-knappen i GUI-versionen blev tildelt denne metode med kommandoen til at ”go north”.

```
@FXML
private void btnNorthEventHandler(ActionEvent event) {
    if (processCommand("go north")) {
        redrawRoom();
    }
}
```

Figur 23: ActionEvent af knappen ”go north”.

Men for at knappen skulle fungere, så blev det oprettet som en onAction-metode, hvor knappen blev tegnet i Scene builder som et FXML. Når man trykkede på knappen, så gav den kommandoen til at spilleren skulle gå mod en retning, men der blev dog ikke tegnet et rum i det næste rum. Derfor blev der tilføjet en metode som hed **redrawRoom()** som tegnede rummene og tilføjede indholdet hver eneste gang man gik ind i et rum. Dette kan ses herunder:

```
private void redrawRoom() {
    removeItems(textUI.getGame().getPlayer().getPreviousRoom());
    removeFire();
    removeSmoke();
    printItems();
    printFire();
    printSmoke();
    printDirectionButtons();
    setBackground();
    drawHealthBar();
    stepCounterText();
    txtAreaNPC.setVisible(false);
}
```

Figur 24: redrawRoom()-metoden tegner alt indhold i de bestemte rumme.

Brugergrænseflade

Første iteration

I 1. iteration blev der tildelt koden til World of Zuul, hvor man skulle viderearbejde spillet. Spillet var dog designet som en command-line interface, hvor det hele foregår via. en slags konsol. Kravet var at designe sit eget spil ud fra World of Zuul, hvor spillet er opbygget af World of Zuul og har dens karakteristika. Spillet i 1. iteration er derfor en command-line interface, hvor input er det indtastede kommandoer, der får spillet til at udføre forskellige bevægelser som er outputtet.

Anden iteration

Til forskel fra CLI-versionen er der i GUI-versionen fokuseret mere på hvordan spilleren igennem den visuelle del, ser spillet som en del af sin læring. Det visuelle element gør at man kan bruge grafiske elementer i den læring som vi ønsker at give til bruger, hvorimod at man i CLI'en var tvunget til at forklare hvert element i tekst.

Den grafiske brugergrænseflade har også gjort det muligt for gruppen at tilføje knapper som gør bevægelsen igennem rum væsentligt mere brugervenligt, og følelsen af at interagere med et objekt er blevet mere virkeligt af at kunne klikke på et billede af objektet frem for at skulle skrive en kommando med navnet på objektet. "Take FireEX"

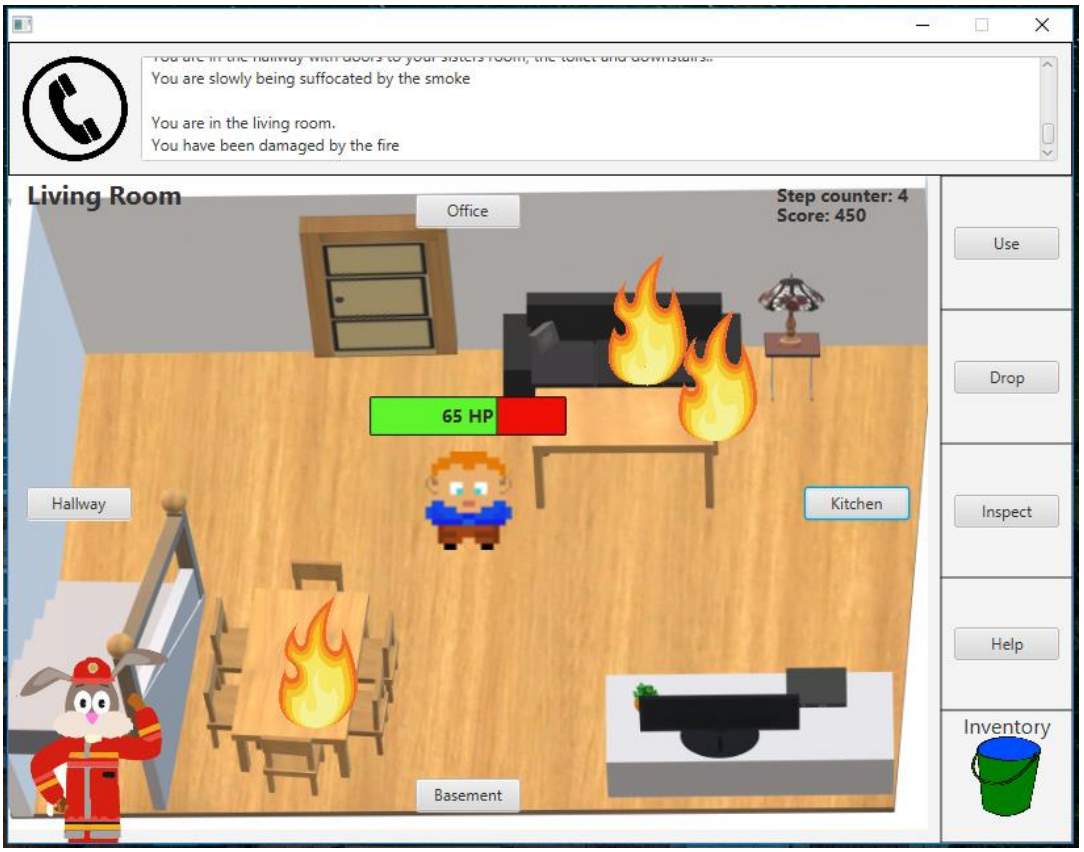
Det at kunne være i stand til at kunne lave en grafisk brugergrænseflade har været med til at kunne give spillet et mere moderne præg, da den tidligere Command line interface fra 1.iteration hører til et element man tidligere var nødsaget til at bruge for at få oversat vores kommandoer til et sprog som computere kunne forstå.

GUI-versionen er bygget efter en fast vinduestørrelse for at sikre at brugeren får den korrekte oplevelse af de grafiske elementer som er medtaget i projektet, da man ikke ønsker en brugergrænseflade hvor tekst eller billeder ikke er læselige. I projektet er der sat fokus på at bruge JavaFX og Java FXML, da dette er de 2 former for GUI der er blevet arbejdet med i programmeringsfaget, og dermed er dem gruppen har bedst kendskab til, dog findes der mange andre måder at lave grafiske brugerflader i Java.

Rigtig mange nyere spil bruger ***WASD***, eller ***Piletasterne*** til at flytte spilleren rundt på den pågældende bane, men i spillet er der bevidst valgt at lave det så spilleren flytter sig rundt i rummene ved hjælp af knapper.

Dette er ikke blot nemmere, det gør også at spilleren laver en bevidst handling om at gå fra et rum til et andet, og dermed sikre sig at spilleren strategisk vælger sin vej igennem spillenes rum.

Siden grafiske brugergrænseflader blev udviklet til computere, har det åbnet nye døre for hvem der kan bruge computere. De grafiske elementer med knapper der kan trykkes på, gør det mindre kompliceret at skulle bevæge sig igennem grænsefladen, og dermed kan en større målgruppe med mindre forståelse for kommandoer også bruge grænsefladen.



Figur 25: Living Room med niveau 1 ild.

Generelt i målgruppen er der gjort brug af forskellige læringsmetoder for at sikre den bedste indlæring, en grafisk brugergrænseflade er den langt nemmeste måde at ramme dem som lærer bedst visuelt igennem læsning. Som ekstra element giver GUI-versionen også en relativ simpel måde at inkorporere lyd sådan at de auditive lærende også får mest muligt ud af spillets vigtige elementer, sammen med de auditive lærende, hjælper lyd og billeder også på dem der skulle have indlæringsvanskeligheder eller ordblindhed.
Den grafiske brugergrænseflade sikrer at spillet formidler den korrekte forståelse og overlader mindre til fantasien, i modsætning til den kommandobaserede brugergrænseflade, hvor alt er påtvunget en længere tekst forklaring.

Java FXML-controller

Til at hjælpe gruppen med den grafiske del af anden iteration, brugte gruppen Scene Builder til at lave en FXML fil som kunne importeres i Netbeans IDE. Netbeans IDE kunne herefter lave en Controller til at styre elementerne i FXML-filen. Brugen af FXML gjorde det muligt for gruppen at separere logikken i det grafiske med logikken den resterende Java kode. Da gruppen havde to FXML-dokumenter blev der oprettet to controllere ved navn **StartMenuController** og **MainController**. Controllerne bestod af en række elementer som opdelte logikken i koden og hjalp gruppen i at opsætte den korrekte logik for hvornår de forskellige metoder skulle køres.

Hver controller lavede en reference til FXML-filens logik, som gjorde at gruppen kunne bruge den viden som lå i filen. Som vist i figuren neden under.

```
@FXML
private Pane paneRoom;
@FXML
private ImageView imgInventory;
@FXML
private Button btnUse;
@FXML
private Rectangle greenbar;
```

Figur 26: Her kaldes de bestemte indhold i FXML-dokumentet

Controllerne består også af en "**initialize**" metode, som kører en given logik lige så snart Controlleren bliver hentet. Gruppen kunne heri sætte de metoder som de ønskede kørt. I **StartMenuController**en blev "**initialize**" brugt til at oprette klassen **TextUI**, dette gjorde at gruppen kunne lave en reference til programmets domænelag. **Initialize** i **StartMenuController**en kan ses i figuren nedenunder.

```
@Override
public void initialize(URL url, ResourceBundle rb) {
    textUI = new TextUI();
}
```

Figur 27: Initialize kører de pågældende controllers

Gruppen brugte controlleren til at lave knapper som spilleren kunne interagere med. Til at køre logikken når knappen blev trykket, brugte gruppen en **EventHandler** som indeholdte de metoder som var krævet af funktionaliteten i den pågældende knap. Eksempel på en **EventHandler** kan ses i figuren nedenfor.

```
@FXML
private void btnEastEventHandler(ActionEvent event) {
    if (processCommand("go east")) {
        redrawRoom();
    }
}
```

Figur 28: ActionEvent af knappen "go east"

Denne **EventHandler** stod for at spilleren rykkede sig et rum til højre når den blev trykket. Samtidig havde den metoden **redrawRoom()** som sørgede for at vise billederne tilhørende det nye rum, og gemme de viste fra det gamle rum.

Controllere er Java filer, og kan dermed oprette metoder ligesom andre Java filer. Dette udnyttede gruppen til at lave metoder som var relevante for at sikre logik blev kørt på det korrekte tidspunkt, eller at hente logik fra domænelaget. I figuren nedenfor ses et eksempel på **highscore()** metoden, som stod for at kalde andre metoder som udregnede spillerens highscore og gemme den til fil.

```
public void highscore() {
    textUI.getGame().getPlayer().setPlayerScore();
    textUI.getGame().saveHighscore();
}
```

Figur 29: Highscore metoden får fat på spillerens score og gemmer score ind i highscore filen.

Værktøj

I hele dette projektforløb blev der indblandet mange former for værktøjer til at kunne færdiggøre produktet. Disse værktøjer består blandt andet af programmeringsværktøjer og tegneværktøjer.

Første Iteration

I 1. iteration var opgaven kun at arbejde på ens spil idé og programmere spillet i form af World of Zuul. Der blev derfor uddelt et tekstbaseret World of Zuul spil som man kunne redigere på og tilføje koder der tilpasses ens spille idé. Det eneste værktøj der blev brugt i 1. iteration var derfor NetBeans IDE.

NetBeans IDE

NetBeans IDE er et ”Integrated Development Environment” program der bruger programmeringssproget Java. NetBeans IDE kan ikke kun bruges til Java programmering, men også forskellige former for programmeringssprog som HTML, JavaScript og CSS.

I projektet blev der brugt NetBeans IDE til at programmere i Java for at lave et tekstbaseret bruger grænseflade af World of Zuul med gruppens idé.

Anden iteration

I 2. iteration blev der indblandet flere værktøjer for at kunne udføre opgaven om at viderearbejde spillet fra 1. iterationen, hvor det nu skal være i form af en grafisk brugergrænseflade. De værktøjer der blev brugt, var Scene Builder, Paint 3D, Paint.NET.

JavaFX

JavaFX er en pakke i programmeringssproget Java, hvor det kan implementere grafiske elementer i sit program. Dette bliver oftest brugt under udviklingen af et program baseret på en grafisk brugergrænseflade.

I projektet blev der brugt JavaFX når der skulle oprettes simple scener, da simple scener ikke kræver meget kode, og var derfor mere simpelt end FXML, hvor der skulle bruges en controller til at fungere.

Scene Builder

Scene Builder bruges til at danne en grafisk brugergrænseflade af et program ved at oprette et FXML-fil i programmet. Scene Builder bruges kun til grafiske brugergrænseflade i Java, da et FXML-fil oprettes, og FXML er et UML baseret bruger grænseflade af JavaFX.

I projektet blev der valgt at lave en kombination af FXML og JavaFX, da FXML var simplere at bruge til at oprette de mere komplicerede brugergrænseflade, mens blev JavaFX brugt til de simplere.

Paint 3D

Paint 3D er et tegneprogram der følges med i operativsystemet, Windows. Programmet kan bruges til at tegne modeller af møbler og alt man kan komme i tanke om, i forskellige former såsom 3D.

Dette program var en af de programmer gruppen brugte til at tegne alt der skulle bruges til at visualisere spillet i den grafiske brugergrænseflade der var oprettet ud fra JavaFX og Scene Builder.

Paint.NET

Paint.NET er et tegneprogram ligesom Paint 3D, men det er ikke noget der hørte med i operativsystemet, Windows, men noget der kunne findes på nettet. Programmet er ligesom Paint 3D, og kan bruges til at tegne forskellige sager.

Der blev brugt to forskellige tegneprogrammer, da der var forskellige erfaringer i gruppen i forhold til tegneprogrammerne. Der er derfor ikke et alt for kompleks grund til brugen af de to forskellige programmer.

8.6 Test

I udviklingsfasen blev der løbende testet for tilføjede funktioner. Der blev arbejdet på at de forskellige funktioner blev delt ud til gruppemedlemmerne i gruppen, så man kunne arbejde alene eller i par om funktionerne. Når funktionerne var færdig programmeret, ville funktionen blive testet af alle gruppens medlemmer. Fungerede funktionen optimalt, blev den implementeret i programmet. Hele gruppen ville for en sikkerhedsskyld gennemgik koden igen for at sikre at programmet stadig fungerede efter tilføjelsen af den nye funktion. Opstod der problemer under testen ville personerne der skrev funktionen gennemgå koden igen og prøve at rette fejlen så programmet virkede. Når det var rettet, ville der være en gennemgang af funktionen igen, hvor alle fra gruppen gennemgik koden og tester programmet. Der skulle være en brugertest for at kunne konkludere om programmet opfyldte dets formål, krav og fortalte om der var nogen ting der skal forbedres så at formålet blev udfyldt bedst muligt til den bestemte målgruppe.

I 2. iteration af projektet foregik udviklingstesten i udviklingsfasen nogenlunde som i 1. iteration, da de forskellige funktioner blev delt ud i gruppen, hvor der blev arbejdet på de forskellige funktioner. Men i stedet for at arbejde enkeltvis på nogle af funktionerne, så blev der par-programmeret for at hjælpe hinanden til at have en bedre forståelse af koden, og der ville derfor også være en mere jævnt delt indsats fra alle i gruppen. Ellers foregik udviklingsfasen, som det gjorde i 1. iteration, hvor alle arbejdede på funktionalitet som blev testet under udviklingen, og implementeret i programmet når funktionen var færdiggjort, fungerede optimalt og var blevet set og godkendt fra alle i gruppen.

Der blev lavet en udgivelsestest af Frederik Verner Helth og hans gruppe som samlet gik ind og testede spillet. De sagde:

”*Spillet havde en meget stor sværhedsgrad måske for høj men måske også godt i forhold til et konkurrenceelement.”.*

De sagde at teksten ikke var synlig nok når man først kom ind i spillet, overså man hurtigt den tekst der blev printet ud og man glemte at bruge Inspect- og Help- funktionen. En af dem nævnte også at de godt kunne have tænkt sig at man kunne bruge tastaturet til at navigere rundt med.

Det var de konstruktive dele de nævnte men de sagde at spillet var meget simpelt at forstå på grund af de tydelige visualiseringer og dets placeringer. Det var et godt børnespil og hvis sprogbarrieren ikke var et problem.

Udgivelsestest var ikke det eneste test der blevet gennemgået, men også brugertest. Spillet blev brugertestet af flere forskellige personer med forskellige baggrunde.

En af testerne er Henrik Steinmann Heidelbach som er tidligere spiltester hos IO interactive. Han gav konstruktiv feedback på forskellige elementer af spillet, samt forslag på forbedringer af spillet i forhold til målgruppen. Til måden at bevæge sig rundt i spillet på sagde han:

”*Spillet er næsten så simpelt så et barn der forstår engelsk kan finde ud af det*”.

Han påpegede en fejl i skiftet mellem det næstsidste og det sidste rum forsaget af en konvertering fra NetBeans til JAR-filen. Opsamlingen af items og brugen af disse var simpel og ligetil. Henrik havde på forhånd ikke modtaget nogen guide i hvordan han bevægede sig igennem rum eller interagerede med items. Spillet var gennemført efter 6 forsøg og 15-20 minutter.
Som en forbedring for at ramme målgruppen var Henrik enig i at en oversættelse af spillet, ville gavne målgruppens forståelse af spillets elementer.

Delkonklusion

Det var svært for gruppen og få testet spillet efter 1.iteration, da der ikke var mange der kendte til kommando baseret interface. Derfor måtte gruppen stole på deres løbende udviklingstest. 2.iteration åbnede for muligheden for at kunne vise spillet til andre end gruppen selv. På den måde fik de udefrakommende til at teste spillet og give feedback på funktionaliteten i spillet, og hvordan udefrakommende spillere forstod spillets baggrund og mål. Gruppen kunne teste spillerne for deres forståelse for brand i bygninger og hvilke komplikationer disse kunne medføre.

10. Diskussion

Kravsspecifikationen sammenlignes med det færdige produkt, for at kontrollere om produktet opfylder disse og lever op til projektets forventninger. I dette afsnit gennemgås kravsopfyldningen for henholdsvis 1. iteration og 2. iteration af programmet.

10.1 Første iteration

Kravspecifikationen består af ti funktionelle og fem ikke-funktionelle krav. Skemaet herunder giver et overblik over opfyldelsesgraden af de relevante krav til 1. iteration:

Funktionelle krav		Ikke-funktionelle krav	
F1: Brugeren skal kunne bevæge sig mellem spillets forskellige rum.	✓	B1: Spillet skal være brugervenligt for børn fra 8 år og op.	~
F2: Brugeren skal kunne indtaste kommandoer som spillet reagerer på.	✓	B2: Spillet skal have en tilstrækkelig ydeevne.	✓
F3: Spillet skal kunne gennemføres.	✓	B3: Spillet skal være nemt at modificere.	~
F4: Spillet skal indeholde objekter, som man kan interagere med og samle op.	✓	B4: Brugeren skal udfordres i sin problemløsningsevne.	✓
F5: Forhindringer der kan fjernes ved brug af objekter.	✓	B5: Brugeren skal gennem spillet undervises i og bruge sin viden om brandsikkerhed.	~
F6: Spillet skal kunne tabes.	✓		
F7: Spillet skal indeholde et point-system.	×		
F8: Spillet skal indeholde non-player character(s).	×		

Ifølge skemaet opfyldte første iteration af programmet seks ud af de otte funktionelle krav. Krav F7 var ikke opfyldt, da denne funktionalitet ikke nåede at blive udviklet. At kravet ikke blev opfyldt, skyldtes mangel på tid. Selvom kravet ikke blev opfyldt, ville det stadig have været muligt at implementere denne funktionalitet, hvis tiden tillod det.

Det andet funktionelle krav der ikke opfyldtes, omhandler NPC'er. Programmet havde i 1. iteration en introduktion, hvor det skulle forestille, at man snakkede med en 1-1-2-operatør. Dog var der ikke nok interaktion mellem bruger og operatøren i denne samtale til, at den kunne kaldes en NPC. Denne funktionalitet ville også kunne tilføjes hvis tiden tillod det.

Angående de ikke-funktionelle krav, så havde første iteration af programmet fuldstændigt opfyldt to og delvist opfyldt tre. Programmet var i første iteration ikke børnevenligt nok, af forskellige årsager. Programmet var tekstbaseret, og krævede derfor en del læsning fra brugerens side. Dette ville udelukke mange børn, som enten ikke kan læse, eller som er vant til grafiske elementer. Derudover var spillet også meget udfordrende. Nye spillere ville skulle bruge mange forsøg før de gennemfører spillet. At tilføje hjælpende forklaringer og andre midler til spillet kunne gøre det generelt mere brugervenligt. At al teksten blev skrevet på engelsk, gjorde at danske børn ikke kunne bruge det.

Kravet om modificérbarhed var kun delvist opfyldt. Programmet var inddelt i klasser med logiske sammenhænge. Dette var med til at gøre fremtidige modificeringer nemmere. Dog var der ingen tydelig lagdeling af klasserne. Man ville derfor i visse tilfælde være nødsaget til at omskrive hele programmet når en enkelt eller en samling af funktioner skulle erstattes. Koden manglede kommentarer der kunne hjælpe udefrakommende programmører med at forstå koden.

Enkelte steder opfyldtes kravet om at programmet underviser brugeren i brandsikkerhed. F.eks. fik brugeren at vide hvad man skal gøre i tilfældet af brand i spillets introduktion. Dog lærte man ikke ret meget om brandsikkerhed efterfølgende. Programmet var i stand til at undervise brugeren, men ikke i tilstrækkeligt omfang.

Styrker og svagheder i resultatet.

Styrker

- Det var et meget simpelt system som var nemt at forstå.
- Funktionaliteten var separeret i meningsfulde klasser.
- Spillet byggede på en god historie som opbyggede en spændende rød tråd igennem spillet.
- Spillet indeholdte et strategisk element, som tvang spilleren til at følge den logiske vej for at kunne gennemføre spillet.
- Spillet indeholdte generelt en masse funktioner, som arbejdede godt sammen til at forbedre spiloplevelsen.

Svagheder

- Brugergrænsefladen var en svaghed i forhold til spillets målgruppe.
- Spillet henvendte sig til en smal målgruppe pga. dets sværhedsgrad.
- Koden manglede kommentarer for at gøre den nemmere at forstå.
- Systemet havde en relativt rodet struktur. Klasser var ikke opdelt i mapper eller lignende for at samle funktionalitet.
- Programmet afhjalp ikke det overordnede problem ret godt ved at undervise om brandsikkerhed.

Forbedringer

I løbet af 2. iteration skulle programmet tilføjes en grafisk brugergrænseflade. Dog var programmet endnu ikke godt nok til at der udelukkende skulle tilføjes en grafisk brugergrænseflade. For at programmet kom til at opfylde de resterende krav, skulle der tilføjes ekstra funktionalitet, som f.eks. yderligere kommunikation mellem 1-1-2-operatøren og spilleren. Dette kunne gøre spillet bedre til at formidle viden om brandsikkerhed. Derudover ville det hjælpe hvis man var i stand til at vælge en sværhedsgrad før spillet gik i gang eller hvis spillet havde mulighed for at guide brugeren igennem det.

Ren kodemæssigt, kunne systemet med fordel opdeles i flere pakker og lag. Dette ville samle funktionalitet og mindske koblingen mellem programmets forskellige moduler, hvilket er vigtigt, hvis man vil genbruge eller modificere dele af programmet. Derudover skulle kommentarer tilføjes til koden. Det kunne være svært at forstå hvordan de forskellige dele fungerer, hvis man ikke havde været med til at udvikle programmet. Nogle dele af programmet var heller ikke kodet på en god måde. Mange steder ville der være tilbøjelighed til fejl hvis en anden del af koden blev ændret.

For at gøre spillet bedre og mere spændende, kunne man tilføje flere objekter og mere funktionalitet. En funktion som gruppen overvejede at tilføje, var en form for røg. Specifikke rumme skulle indeholde røg som skadede spilleren eller kunne hele banen have lidt røg som gradvist skadede spilleren. Dette ville sørge for, at spilleren ikke bare kunne vandre rundt i al evighed inde i bygningen.

I denne iteration nåede programmet aldrig at få en score-funktion. Denne score skulle være baseret på antallet af skridt det tager at vinde spillet. Eventuelt kunne man også gøre ens score afhængig af hvor meget liv spilleren havde tilbage i slutningen af spillet eller hvilke handlinger spilleren havde taget. Disse scores skal helst gemmes i en fil på computeren. På denne måde var man i stand til at gemme og tilgå tidligere highscores.

Det var begrænset hvor meget brugeren blev undervist i brandsikkerhed ved at anvende dette program. Undervisning i brandsikkerhed har været formålet med dette projekt, så dette krav er en essentiel del af programmet.

10.2 Anden iteration

Efter at have udviklet anden iteration af programmet, opfyldte programmet flere af kravene. Dette ses i nedstående skema som viser 2. iterations opfyldelse af de relevante krav.

Funktionelle krav		Ikke-funktionelle krav	
F1: Brugeren skal kunne bevæge sig mellem spillets forskellige rum.	✓	B1: Spillet skal være brugervenligt for børn fra 8 år og op.	✓
F3: Spillet skal kunne gennemføres.	✓	B2: Spillet skal have en tilstrækkelig ydeevne.	✓
F4: Spillet skal indeholde objekter, som man kan interagere med og samle op.	✓	B3: Spillet skal være nemt at modificere.	✓
F5: Forhindringer der kan fjernes ved brug af objekter	✓	B4: Brugeren skal udfordres i sin problemløsningsevne.	✓
F6: Spillet skal kunne tabes.	✓	B5: Brugeren skal gennem spillet undervises i og bruge sin viden om brandsikkerhed.	~
F7: Spillet skal indeholde et point-system.	✓		
F8: Spillet skal indeholde non-player character(s).	~		
F9: Spillet skal have en grafisk brugergrænseflade.	✓		
F10: Brugeren skal kunne interagere og navigere rundt i spillet ved hjælp af grafiske elementer.	✓		

De to funktionelle krav der ikke blev opfyldt af 1. iteration var nu henholdsvis opfyldt og delvist opfyldt. Igennem 2. iteration fik programmet tilføjet et point-system, hvilket opfyldte krav nummer F7. I 2. iteration blev en form for NPC tilføjet i form af den hjælpende brandmand. Denne karakter var i stand til at sige forskellige sætninger baseret på hvor langt brugeren er noget. Dog var det begrænset hvor meget brugeren var i stand til at interagere med den, og kravet blev derfor kun delvist opfyldt.

Kravene F9 og F10 var eksklusive til anden iteration af programmet. Disse blev også opfyldt, da et nyt vindue blev åbnet når programmet blev startet. Brugeren havde nu heller ikke mulighed for at indtaste kommandoer, men blev tvunget til at navigere rundt i spillet ved hjælp af de indbyggede knapper.

Programmet var også blevet endnu mere brugervenligt siden sidste iteration. Sværhedsgraden var ikke ændret, så programmet ville stadig være udfordrende for nye spilleren. Dog blev der tilføjet et hjælpende element i from af brandmands-NPC'en. Han kom med hjælpende sætninger der kunne guide brugeren igennem spillet. Programmets brugervenlighed blev også styrket af den grafiske brugergrænseflade, da brugerne nu ville slippe for at læse lange tekster, men bare kunne se på billederne.

Programmet blev også opdelt i flere pakker og lag. Denne opdeling har gjort det nemmere for fremtidige udviklere at forstå og udskifte elementer i spillet. Lagdeling tillader også mere problemfri genbrug af kode. Et andet aspekt der hjalp til programmets modificeringsmuligheder, var tilføjelsen af kommentarer. Alle klasser indeholdte efter anden iteration en forklarende tekst om klassen og alle dens metoder.

Det ikke-funktionelle krav nummer B5 var efter anden iteration stadig delvist opfyldt. Siden første iteration blev der ikke tilføjet markante funktionaliteter der havde til opgave at undervise brugeren i brandsikkerhed. Det eneste brandsikkerhedsmæssige som programmet indeholdte var samtalen med 1-1-2-operatøren og samtalen med brandmands-NPC'en. Hele programmet var stadig på engelsk, så danske børn ville ikke kunne bruge det.

Styrker og svagheder ved programmet

Styrker

Efter anden iteration havde programmet alle de samme styrker som første iteration havde. Programmets nye styrker inkluderede:

- Lagdelt funktionalitet gav adgang til nem modificering og genbrug.
- Grafisk brugergrænseflade henvendte sig til målgruppen.
- Forklarende kommentarer i hele koden.

Svagheder

Mange af de svagheder som gjaldt i første iteration, var efter anden iteration fjernet. Dog nåede alle svaghederne ikke at blive rettet i anden iteration:

- Brugervenligheden var stadig svækket af den hårde sværhedsgrad.
- Programmet afhjalp ikke det overordnede problem ret godt ved at undervise om brandsikkerhed.

Forbedringer

Efter anden iteration blev udviklet, blev mange af manglerne og fejlene fra første iteration rettet. Med den begrænsede tid var det dog ikke muligt at helt eliminere alle forbedringsmuligheder. Efter anden iteration, manglede programmet stadig fuldstændigt at opfylde de sidste krav. Kravet om en NPC kunne opfyldes ved at tilføje flere muligheder ved kommunikationen mellem bruger og brandvæsen. En anden mulighed var at føje personer til spillets rum som brugeren kunne hjælpe ud. Dette ville skabe mere fylde i spillet. Dog var det ikke et stort problem at kravet om NPC'er ikke blev opfyldt, da kravet havde en lav prioritet.

Ud over den manglende interaktion mellem bruger og NPC, havde programmet heller ikke fået nogle ekstra undervisningselementer. Dette var en central del af projektet, men gruppen valgte at nedprioritere dette, da de ønskede et fungerende program. Hvis tiden tillod det, kunne kravet godt opfyldes. Gruppen overvejede at tilføje detaljerede beskrivelser af spillets brandudstyr som kunne blive vist til brugeren, når det blev samlet op.

Delkonklusion

Igennem første iteration fik gruppen skabt et program som opfyldte en del af de opstillede krav. Samtidigt med at brugergrænsefladen skulle omdannes til grafisk, blev flere funktioner tilføjet. Dette sikrede, at det endelige produkt opfyldte alle kravene og nogle delvist. At et par krav endte med ikke at blive fuldstændigt opfyldt skyldtes en mangel på tid. Hvis projektet strakte sig over en længere periode, ville disse krav uden tvivl kunne opfyldes.

11. Konklusion

Igennem første del af projektet fik gruppen fremstillet et produkt som anvendte en tekstbaseret brugergrænseflade. Programmet udspillede et scenarie hvor man befandt sig i en brændende bygning og skulle slippe ud. Allerede dér var størstedelen af programmets krav opfyldt. Igennem anden iteration lykkedes det at få omdannet brugergrænsefladen til grafisk. Samtidigt blev mere funktionalitet tilføjet som sikrede, at resten af kravene blev opfyldt og delvist opfyldt. Gruppens projekt endte med at have et produkt, som de generelt var tilfredse med. Det kunne køres, spilles, gennemføres og opfyldte de fleste krav der blev stillet. Nogle krav blev kun delvist opfyldt af programmet, men disse var ikke kritiske. Det betyder ikke, at disse krav var uopnåelige. Hvis projektforløbet strakte sig over længere tid, kunne disse krav godt have været opfyldt.

Igennem diskussionen af programmets kravopfyldelse, kunne nogle af problemformuleringens spørgsmål besvares. Det har vist sig, at det faktisk er muligt at bygge et underholdende og læringsrigt program ud fra World of Zuul-skabelonen som udspiller et scenarie, hvor man befinder sig i en brændende bygning. Det har også lykkedes at lave programmet børnevenligt, selvom det behandler et så skræmmende og seriøst emne som brandsikkerhed. Det har også lykkedes at få implementeret et konkurrenceelement i form af en highscore. Det var oprindeligt planlagt, at programmet skulle udlodde en gevinst til dem som gennemførte. Dette blev dog aldrig til noget. På trods af dette, mener gruppen, at projektet har kunne bekræfte problemformulerings hovedspørgsmål, da det har lykkedes at fremstille et læringsspil til børn og unge der kan bidrage til deres viden om brandsikkerhed.

12. Perspektivering

Gruppen måtte efter 1. iteration erkende at lektier i kodningen ikke var optimalt. Kodningen burde i stedet være lavet i fællesskab for at skabe en bedre fællesforståelse af emnet og dets tekniske aspekter. Dette var en læring som blev taget med i 2.iteration.

Grundet gruppens nylige introduktion til Github, kendte de ikke til funktionaliteten med at tilføje medprogrammører når man tilføjede kode. Derfor blev deres Github-insight ikke særlig præcis, hvad angår antallet tilføjede linjer.

Gruppen havde pga. deres undervisningsrækkefølge ikke lavet den bedste proces i projektet, da de manglede viden om de ting de skulle bruge for at lave den mest velovervejede kode. De havde ikke overvejet hvilken form for softwareprocesmodel som blev benyttet igennem projektet.

I 2. iteration modtog gruppen ikke nok viden om kodning i GUI, og brugte derfor den første uge på at få en forståelse for kodningen. Det formindskede tidsmængden for gruppen til at kunne opfylde alle krav til spillet. Gruppen nåede i 2. iteration ikke at opfylde kravene såsom at gøre spillet mere lærerigt, muligt at vælge sværhedsgrad, oversættelse fra engelsk til dansk og brugertest på målgruppen.

Gruppens problemformulering har været god igennem hele projektforløbet, da den indeholder et problem der har relevans for størstedelen af Danmarks befolkning. Hvis gruppen skulle starte projektet forfra, ville problemformuleringen blive beholdt. Men starten af udviklingen af programmet ville blive behandlet anderledes. Ved at starte ud med helt ny viden om programmeringen, gjorde det at gruppen havde en rodet kode uden et fastlagt design. At have lavet designet på forhånd kunne derfor have effektiviseret selve implementeringen.

13. Litteraturliste

Beredskabsstyrelsen. (29. september 2017). *Uge 40 - Fokus på brandforebyggelse*. Hentet fra Beredskabsstyrelsen: brs.dk/forebyggelse/brand/kampagner/uge-40-fokus-paa-brandforebyggelse/Pages/uge-40-fokus-på-brandforebyggelse.aspx

Beredskabsstyrelsen. (06. juli 2018). *Dødsbrande og omkomne ved brand* . Hentet fra Beredskabsstyrelsen: http://brs.dk/viden/statistik/doedsbrande/Pages/doedsbrande.aspx

Beredskabsstyrelsen. (01. oktober 2018). *Omkomne i brand*. Hentet fra Redningsberedskabets statistik: https://statistikbank.brs.dk/sb#page=95457503-60d1-4a22-9cee-dd78bb0aa8d0

Danske Beredskaber. (07. september 2017). *Flere bør tale med deres børn om brand*. Hentet fra Danske Beredskaber: https://danskeberedskaber.dk/flere-boer-tale-deres-boern-brand/

Ebdrup, N. (06. januar 2015). *Vi bliver motiverede, bare af at tro vi spiller et spil*. Hentet fra Videnskab.dk: https://videnskab.dk/kultur-samfund/vi-bliver-motiverede-bare-af-tro-vi-spiller-et-spil

IDA. (05. september 2017). *Ekspert: Positivt at forældre taler med børn om brand*. Hentet fra IDA.dk: https://ida.dk/content/ekspert-positivt-foraeldre-taler-med-boern-om-brand

Jensen, K. R. (25. juli 2018). *Professor: Ekstreme skovbrande bliver mere almindelige*. Hentet fra DR Nyheder: https://www.dr.dk/nyheder/viden/klima/professor-ekstreme-skovbrande-bliver-mere-almindelige

Sander, N. (10. juli 2018). *Tørke sætter Danmark i flammer: Over 1.000 naturbrande siden 1. maj*. Hentet fra DR Nyheder: https://www.dr.dk/nyheder/indland/toerke-saetter-danmark-i-flammer-over-1000-naturbrande-siden-1-maj

II. Procesrapport

1. Fælles forventninger

For at gruppen kunne arbejde ensartet igennem projektet blev de nød til at have fælles forventninger til målet med projektet. Gruppen blev enige om at fagligheden var en af de højest prioriterede emner, da alle gruppens medlemmer var uden meget eller helt uden tidligere programmeringserfaring, og derved blev der aftalt at skreven kode blev gennemgået på gruppen, så spørgsmål der kunne opstå undervejs ville blive taget når koden var frisk. Dette gav også mulighed for at få de andre gruppemedlemmers syn på metoderne, så en eventuel smartere metode kunne blive forslået.
Det var vigtigt for gruppen at det færdige produkt var noget som alle i gruppen var tilfredse med, så eventuelle metoder og funktionaliteter som gruppens medlemmer var splittede om skulle implementeres, ville være løst inden det endelige produkt blev afleveret. Dette var også vigtigt for at gruppen alle kunne stå inden for den kode som projektet indeholdte.
Det var vigtigt for gruppen at alle gjorde det bedste de kunne, men på deres eget niveau, sådan at de med samarbejde kunne opnå et tilfredsstillende resultat. Det var vigtigt at alle gruppens medlemmer var opmærksomme på at der var forskellige niveauer, og det derfor ikke var alle gruppens medlemmer som var lige så ”dygtige” programmører som andre. Dette skabte et læringsmiljø som gjorde at gruppens medlemmer brugte tid på at hjælpe hinanden så de den vej igennem kunne opnå et højt gennemsnitligt niveau.
For at kunne overholde deadlines blev det aftalt at alle gruppens medlemmer fuldførte de opgaver de var blevet stillet bedst muligt, og derfor også selv læste korrektur på den stillede opgave, for bedst muligt at mindste eventuelle fejl der skulle rettes senere af resten af gruppen.

2. Motivation

Gruppens medlemmer skiftes til at give deres motiverende faktor for gennemførsel af projektet.
Disse faktorer opsummeret til vigtigheden af at man følte et ansvar overfor gruppen og den afsluttende karakter. Disse faktorer var for gruppen vigtige at have med for at have noget at bruge i tilfælde af at gejsten i gruppen pludselig skulle falde, så de med faktorernes hjælp kunne genvinde motivationen og arbejdet kunne fortsætte.
Gruppens sammenhold og lærelyst har alle medvirket til en god gruppedynamik, som gruppen har draget stor nytte af. Det har medvirket til at gruppen har kunne lave fælles aftaler om arbejde og mødetider for dermed at finde frem til løsninger som passede alle gruppens medlemmer. Samtidig har alle gruppens medlemmer kunne udtrykke sig frit hvis der var nogen af de resterende medlemmers meninger som de ikke var enige i, dette har resulteret i giverige debatter som har hjulpet gruppen til at finde frem til den mest optimale vej igennem projektet.

3. Samarbejdet med vejleder Lone Borgeren

Muligheden for at kunne stille utallige spørgsmål til vejlederen, har været af stor betydning for at gruppen for at kunne holdet projektet på rette spor igennem forløbet. Ligeledes har vejlederen hjulpet gruppen med løbende bidrag til de, af gruppen i forvejen igangværende processer.
Tidsrummet mellem vejledermøderne har passet med at gruppen har kunne bygge videre på projektet, for derefter at fremlægge resultatet for vejlederen som bidrog med rådgivning.

4. Problemorienteret projektarbejde

Det faktum at gruppen havde frie tøjler til at finde frem til det problem som skulle blive udgangspunktet for gruppens projekt, var en vanskelig proces, i det at gruppen var fastsat på at problemet skulle være reelt. De frie tøjler gjorde samtidig at igennem idégenereringen kunne gruppen gennemgå idéerne og vælge den gruppen fandt mest interessant. Problemet brødfødte en masse problemstillinger som der kunne arbejdes videre med, og ressourcepersoner med viden omkring problemet blev kontaktet for at opnå baggrundsviden om projektet.

Gruppen aftalte at der igennem projektets forløb løbende blev aftalt hvor meget individuelt og gruppe arbejde der ville blive lavet, men med fokus på at der ikke måtte laves elementer som ikke blev præsenteret for resten af gruppens medlemmer, for derfor at opretholde en fælles forståelse for kodens indhold.
Gruppen sørgede løbende for at prioritere de idéer der lå, så gruppen kunne sikre at de vigtige elementer i projektet, var færdige til deadline. Denne prioritering gjorde at der var funktionaliteter som gruppen ikke fik færdige for at prioritere vigtigere elementer.

Morten Jensens projektarbejde

Læring og refleksion

Morten fik meget ud af at kombinere objektorienteret programmering. Ligeledes fik han meget ud af at kombinere objektorienteret programmering og introduktion til software Engineering. Gruppens måde at håndtere den proces det var at lave programmet og rapporten, fungerede godt da alle medvirkede og ingen dermed blev overvældet af arbejdet.
Da Morten startede på projektet var han uden programmerings erfaring, men efterhånden som projektet og undervisningen skred frem, udviklede hans kompetencer sig, så han på egen hånd kunne udvikle kode på som var relevant for projektet. Projektet har givet Morten en væsentlig forbedret forståelse af problemorienteret projektarbejde, og hvordan man igennem dette kunne udvikle et produkt der var løsningsrelevant.

Projektet har givet mulighed for at kunne arbejde selvstændigt uden at være bange for at støde på fejl, da Morten mente han havde gruppens opbakning, og altid kunne finde hjælp i gruppens medlemmer, eller vejleder.

Projektarbejdet

Morten mener at der på trods af lidt forvirring i projektets opstart har været en god arbejdsindsats fra alle gruppens medlemmer, dette har muliggjort at der lige fra projektets start har været plads til frie tanker med henblik på projektets emne og udvikling af programmets funktioner. Han syntes at projektets opbygning har passet fint til et 1.semester projekt, da han følte forventninger lå realistisk i forhold til de kompetencer gruppens medlemmer udviklede undervejs. Morten forsøgte at bidrage så meget som han kunne med konstruktive løsninger eller ændringer i forhold til problemet og produktet. Morten var glad projektets type, da det at skulle løse en problemstilling med et produkt, på ingen måde var urealistisk i forhold til hvad der sker i mange firmaer. Dette var med til at han kunne holde motivationen igennem projektet.
Morten var enig i gruppens holdning til fælles læring, det dette gav mulighed for gruppens medlemmer at opnå et mere ligeligt niveau i slutningen af projektet. Han syntes projektets resultat var en god afspejling af gruppemedlemmerne kompetencer, og han er glad for produktets form.

Thomas Steenfeldts projektarbejde

Læring og refleksion

I starten af projektet satte Thomas sig nogle mål for hvad han ville opnå igennem projektarbejdet. Hans personlige mål lød på, at han ville få skabt ny viden og løsninger og måske endda hjælpe andre mennesker med at få skabt en ny viden. Igennem projektet har Thomas fået skabt stor viden omkring mange forskellige emner. Igennem projektet har vi fået programmeret meget, og ved at eksperimentere og sidde med det selv, lærte Thomas faktisk rigtig meget ud over hvad man lærte igennem OOP-undervisningen. Derudover er Thomas også blevet klogere på faget Introduktion til Software Engineering og vigtigheden i at følge dets principper. Thomas har også fået en bedre forståelse for generelt projektarbejde. I starten var han ikke helt sikker på, hvordan det egentlig kom til at foregå på denne uddannelse, men efter dette projekt, har han lært, hvad projektarbejdet egentligt fungerer.

Projektet har bestået af mange forskellige forløb og gruppen har arbejdet på mange forskellige former igennem det. Nogle gange sad man individuelt og programmerede, og nogle gange sad sammen og parprogrammerede. Begge metoder har været med til at bidrage til Thomas' læring. Dog har den mest effektive metode været den hvor man individuelt sidder og arbejder. Her får man lov til at fordybe sig i et emne og sidde med sine egen tanker. Ved programmering i hold, bliver man ofte distraheret. Dog kan der være fordele ved hold. F.eks. kan man få flere vinkler på emnet.

Projektarbejde

Thomas mener, at projektarbejdet har fungeret rigtigt godt. Igennem hele forløbet har alle bidraget til arbejdet. Det er umuligt for alle at bidrage lige meget til projektet pga. forskellen i faglige kompetencer, men det gør heller ikke noget, da Thomas mener, at alle har haft den samme tankegang om projektets vigtighed. Der var ingen gruppemedlemmer der var mere ligeglade med projektet end andre.

Thomas' arbejdsindsats til projektet har generelt været god. Under gruppemøderne har han så vidt muligt bidraget til projektet og arbejdet koncentreret. Han har været god til at holde sig på sporet til gruppemøderne. Dog havde han ønsket at bruge lidt mere tid på projektet derhjemme. Det var svært at komme i gang med at arbejde derhjemme, da de andre fag tog meget tid og energi, og fordi Thomas ikke har været lige god til at tage sig sammen til at arbejde når han var derhjemme. Derudover skal Thomas' arbejde også fungere mere effektivt. Han bliver nemt distraheret hvis der arbejdes hjemme.

Alexander Nguyens projektarbejde

Læring og refleksion

I starten kom Alexander ind i projektet uden noget viden til programmering. Hans mål var derfor at opnå ny viden og skabe erfaring til programmeringen. Kurset OOP har hjulpet Alexander med at starte ud med at få en forståelse for programmering, og dermed resulteret i mere viden i forhold til programmeringen. Men viden uden erfaring hjælper ikke når man skal kode et system. Alexander har derfor i sin fritid programmeret lidt for at selv skabe erfaring til programmeringen, men semesterprojektet har hjulpet en stor del med at skabe erfaring og skabe ny viden indenfor programmering. Programmeringen består ikke kun af programmering, men også alt andet, som læres gennem kurset Introduktion til Software Engineering som har vist Alexander at der er meget mere til programmering og hans studie.

Semesterprojektet er blevet brugt som en sammensat af de forskellige kurser til at skabe mere viden og erfaring, og har derfor hjulpet Alexander med at opnå hans mål. Han har blandt andet forstået at programmering alene er ikke altid optimalt, da der kan altid være noget man ikke forstår. Parprogrammering introducerede ham for en ny metode for at skabe nyt viden og erfaring indenfor programmeringen, da man deler og modtager viden og erfaring fra andre.

Projektarbejde

Projektarbejdet gik virkelig godt, da der har været et godt samarbejde i projektgruppen. Alle har haft et godt samarbejde med hinanden, og prøvet at bidrage ligeligt selvom alle har forskellige erfaringer i forhold til opgaverne i projektet. Alexander synes at alle har været gode til at diskutere om de forskellige muligheder til projektet og været gode til at modtage hinandens bidrag.

Alexander mener selv at han har bidraget godt til projektet, og har prøvet at dele sin viden og erfaring med andre fra gruppen og samtidig modtage. Der har været få forvirringer i gruppen, men alle har været gode til at løse forvirringerne. Der er blevet arbejdet koncentreret og samtidig prøvet at holde det hyggeligt at arbejde på projektet sammen. Alexander mener derfor at det har medført til at alle har prøvet deres bedste til at bidrage og til at forstå om programmering. Alle fra gruppen har prøvet at udvide deres viden indenfor programmering, men der har været forhindringer fra andre kurser, som medførte til at ikke alle kunne bruge alt for meget af deres tid derhjemme til at arbejde på projektet.

Jacob Wowk Jørgensens projektarbejde

Læring og refleksion

I starten kom Jacob ind projektet med erfaring i front-end kodning i hjemmesider og at arbejde med grafiske redigeringsværktøjer. Hans personlige mål var at han gerne ville skabe ny viden og forståelse inden for programmering. Der har Objekt orienteret programmering kurset været en stor hjælp, han lærte en del på at kode hjemmefra i form af lektier. Men Jacob lærte det bedst af alt når gruppen parprogrammerede eller gennemgik koden hvor gruppens medlemmer indgik i en diskussion. Ud over programmering ville han også gerne forstå teorien bag systemudviklingen og dets processer, der har kurset Introduktion til software engineering hjulpet rigtig meget. De mange journaler og dets vurderinger har hjulpet gevaldigt på forståelsen samt hvordan man skal dokumentere forholdsvis komplekse begreber. Gruppens vejledermøder med Lone har virkelig givet en kæmpe forståelse på mange af de teoretiske fagbegreber.
Begge disse dele har virkelig åbnet øjnene for hvad denne uddannelse går ud på.

semesterprojekt har været samlingspunktet for de vigtigste dele af den undervisnings han har haft. Projektet er en kombination af de tekniske elementer fra Objekt orienteret programmering kurset og de teoretiske elementer fra Introduktion til software engineering kurset. Dette har gjord at han forstår de tekniske elementer meget godt, men har stadig en smule bøvl med at implementere kode det da han stadig godt kan side fast af og til, som gør at han ikke er helt selv sikker nok til at redigere kode uden at gennemgå det med gruppen. derfor havde han en smule svært ved at

Projektarbejde

Jacob mener at projektarbejdet fungerede rigtig godt imellem hvert medlem af gruppen. Han syntes alle havde en god kommunikation, godt humør og haft enighed om forventningen til hinanden og projektet. Han syntes de dog at koden endte med at gå lidt hurtigt da der blev lavet en stor del på 3 dage udenfor et aftalt mødetidspunkt, hvor det ikke var muligt for ham at møde op. Det fik ham til at føle at han bidragede en smule mindre i koden end de andre. Han fik dog indhentet det efterfølgende I parprogrammeringen, gruppen var desværre ikke særlig opmærksomme på at sætte de medlemmer på som bidragede til koden inden der blev commitet. Men han føler dog at han fik kommet igen og lavet det mere efterfølgende. alle i gruppen havde det super godt sammen og dette medførte bare et bedre resultat, da man har lyst til at tage i skole med en god gruppe.

5. Tidsplan

	Opstart					Fri	1. Iteration				2. Iteration					Fri
opgave/uge	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
samarbejdaftale	x	x														
vejlederaftale	x	x														
Idégenerering		x	x	x	x	E	x	x			x					J
problemanalyse			x	x	x	F										U
problemformulering			x	x	x	T										L
postersession					x	E										E
Projekt grundlag			x	x	x	R										F
gennemgang af World Of Zuul						Å	x									E
Kravspecifikation						R	x	x			x					R
konceptudvikling						S	x	x			x					I
Programmering						F		x	x	x	x	x	x	x		E
uml						E			x	x			x	x	x	
midtvejsseminar						R				x						
1 iterations udkast						I				x						
processforbedringer						E					x					
Rapportskrivning														x	x	
Aflevering af spillet														x		

Figur 30: Egentlig tidsplan

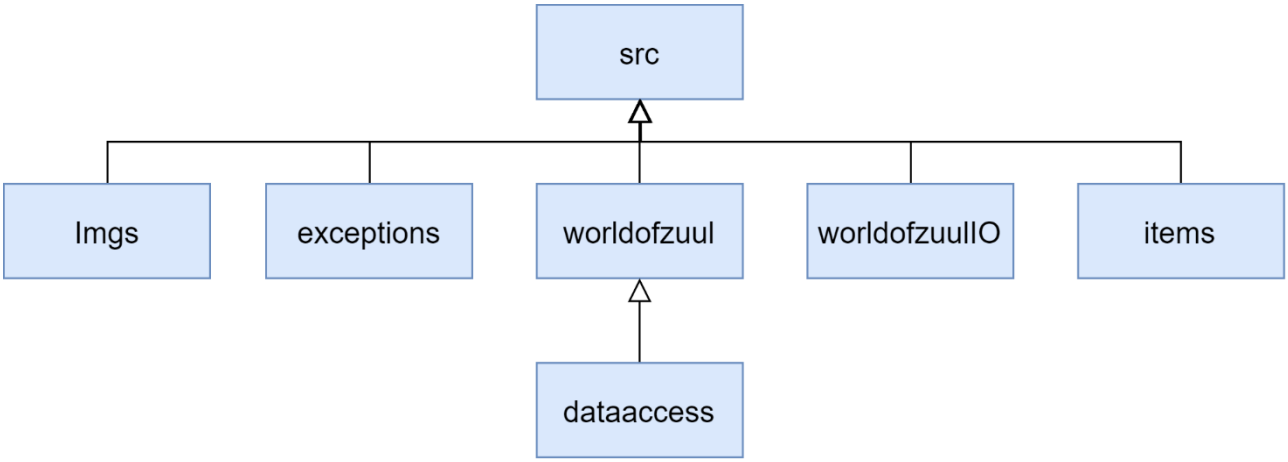
I starten af projektet blev der lavet en forventet tidsplan over hele projektforløbet.
Den blev ikke overholdt helt, derfor blev der lavet en specifik tidsplan for forløbet.
Der ses på billedet hvordan det endte med at arbejdet blev fordelt på tiden som har været til rådighed.
Gruppen havde en smule fejl med vurderingen af tid der skulle bruges på hvert element.
Nogen af tingende blev lavet hvor der efterfølgende kom undervisning i emnet som var relevant, som der ikke var benyttet.

Rapportdelen i anden iteration blev ikke overholdt særlig godt, dette var et valg da der blev prioriteret på programmering af funktionaliter og rettelser i koden. Klassediagrammet blev brugt forkert, da det var noget af det sidste der var færdigt i hver iteration, den skulle være lavet i starten og struktureret programmet ud fra den.

III. Oversigt over kildekode

1 Filstruktur

Figuren herunder viser filstrukturen af vores Java-applikation:



Figur 31: Filstruktur af java-applikationen

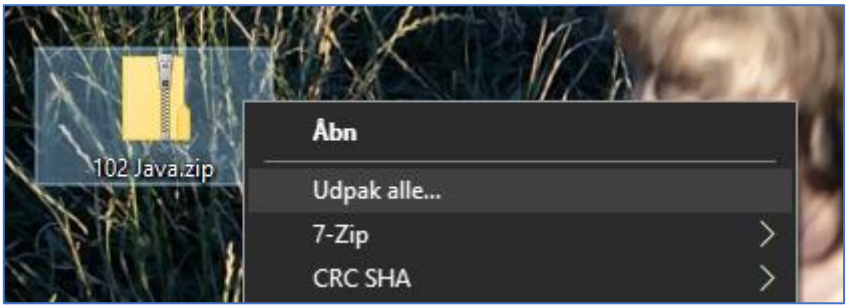
Alle java-klasserne er samlet i mappen **src**. Denne indeholder fem undermapper. Mappen **worldofzuul** indeholder klasserne der udgør programmets domænelag. Derudover indeholder den mappen **dataaccess** som indeholder klasserne der udgør programmets datalag. Mappen **worldofzuulIO** besidder alle klasser der udgør præsentationslaget. Derudover indeholder den også fxml-dokumeterne. Mappen **items** indeholder klassen **Item** og alle dens subklasser. Mappen **Imgs** indeholder alle billeder der anvendes i spillet. Mappen **exceptions** indeholder de exceptions gruppen selv har lavet til programmet

IV. Brugervejledning

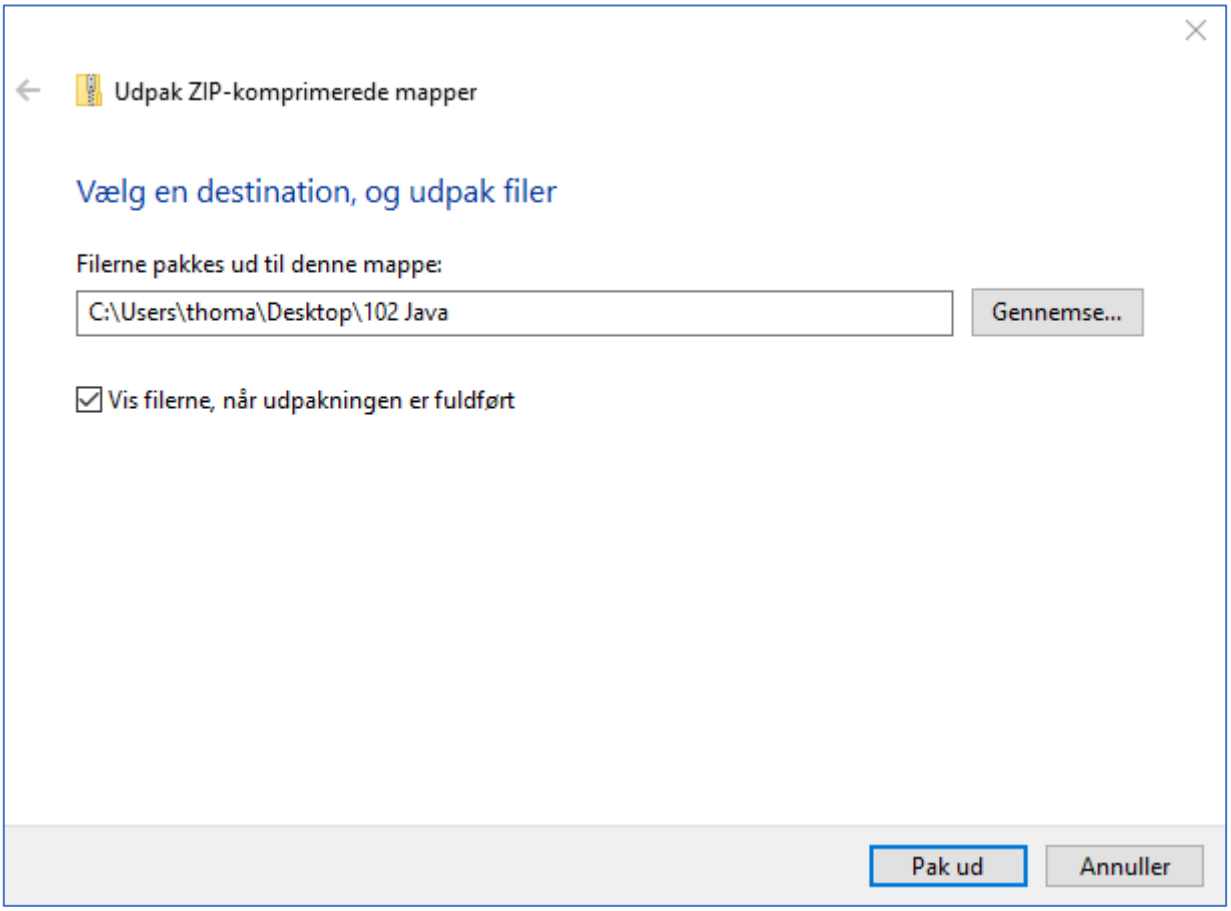
1 Opstart af programmet

I produktafleveringen havde gruppen både afleveret første og anden iteration af programmet. Begge iterationer blev samlet i samme .zip-fil ved navn '102 Java'. Følgende er guide til hvordan enten første eller anden iteration af programmet startes op ved hjælp af NetBeans.

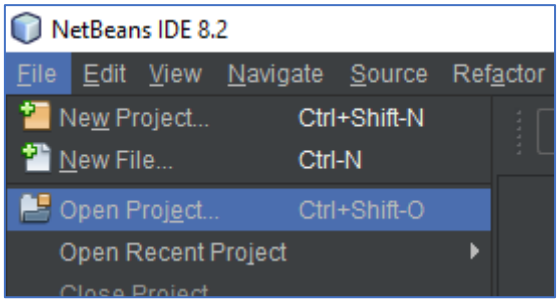
- Højreklik på den downloadede .zip-fil og vælg 'Udpak alle…'



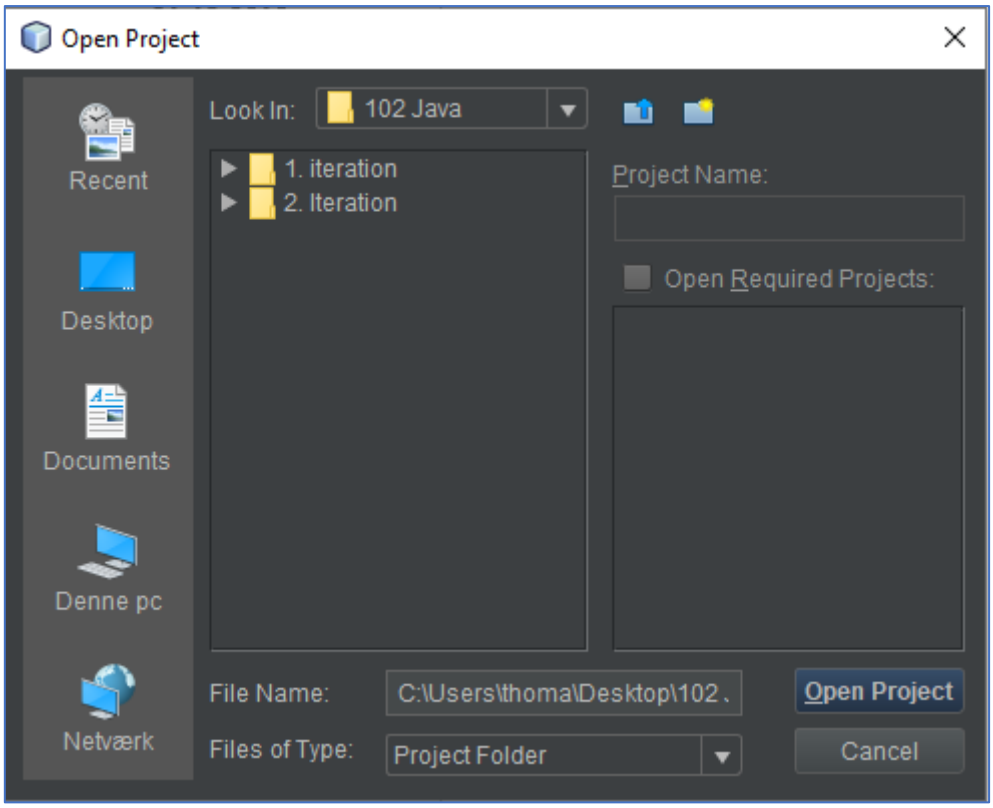
- Vælg udpakningsdestinationen og tryk 'Pak ud'



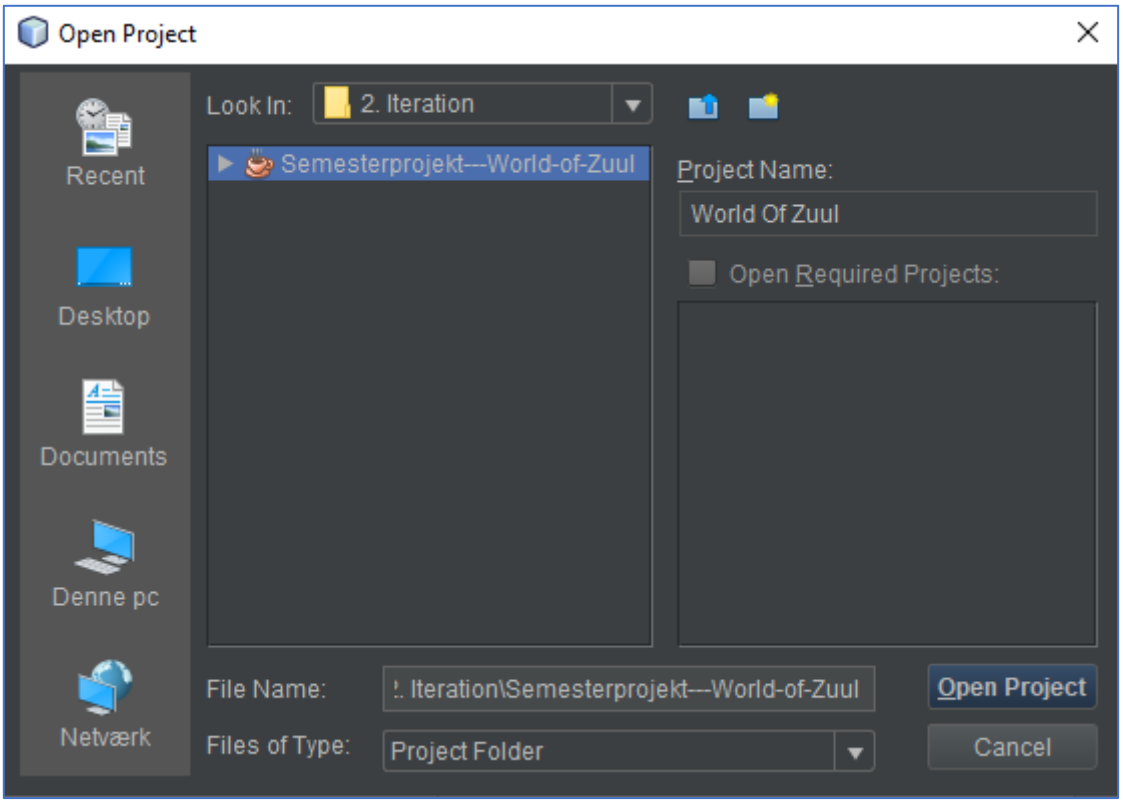
- Åbn Netbeans IDE
- Under fanen 'File' vælg 'Open Project'



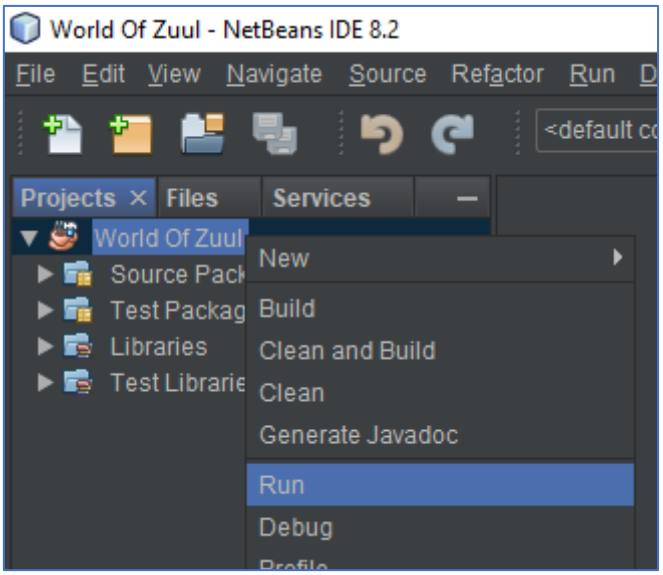
- Navigér til udpakningsdestinationen med den nu udpakkede '102 Java'-mappe. Herinde burde mapperne '1. Iteration' og '2. Iteration' vise sig.



- Åbn den mappe der svarer til det program der ønskes opstartet
- Markér projektet 'Semesterprojekt---World-of-Zuul' og tryk 'Open project'. Det valgte projekt burde nu åbnes i NetBeans.



- Kør projektet ved at højreklikke på det i venstre kolonne under fanen 'Projects' i NetBeans og vælg 'Run'.



- Programmet burde nu starte.

2 Anvendelse af programmet

Følgende guide til anvendelse af programmet er baseret på 2. itereation.

Når programmet startes, vil man blive mødt med en startmenu. Herfra har man tre muligheder: Start, Highscore eller Quit.

2.1 Start

En ny scene dukker op med et tekstfelt og en 'OK'-knap. Brugeren indtaster her sit brugernavn, som senere vil blive brugt til at gemme sin highscore med. Navnet må ikke være tomt eller indeholde kommaer.

Tryk på 'OK' for at fortsætte.

En ny scene viser sig. Her udskrives en introduktion som introducerer spillets handling.

På hvilket som helst tidspunkt kan knappen 'Continue' trykkes på for at starte det egentlige spil.

En ny scene viser sig som nu er brugergrænsefladen til det egentlige spil. Her er en mænge forskellige knapper og tekstfelter.

Rum-vinduet er den centrale del af spillet. Her tegnes det nuværende rum spilleren befinder sig i, spilleren selv, objekter man kan interagere med og knapper til navigation i spillet.

Navigation i spillet

Spillet er opdelt i en mænge rum som spilleren er nødt til at bevæge sig igennem for at gennemføre det.

I rum som har udgange, vil der tegnes knapper med navn på den pågældende udgang. Knappen vil blive placeret i udgangens retning. Hvis der f.eks. var en udgang 'Hallway' mod øst, vil en knap med teksten 'Hallway' blive tegnet i højre side af Rum-vinduet.

Brugeren kan ved hjælp af disse knapper navigere mellem spillets forskellige rum. Ved klik på sådan en knap, vil spilleren blive ført videre til den pågældende udgang. Knapperne kan nu bruges til at navigere videre til yderligere nye rum.

Forhindringer

Igennem spillet vil brugeren møde forskellige forhindringer som hæmmer mulighederne for at gennemføre spillet.

Der er generelt tre forskellige forhindringer i spillet; Ild, røg og låste rum.

Ild

Nogle af de rum spilleren kommer forbi indeholder ild. Når spilleren bevæger sig ind i et rum med ild, vil han tage skade. Mængden af skade afhænger af ildens størrelser. Ild har tre forskellige størrelser; 1, 2 og 3. Skaden udregnes som 25 gange ildens størrelse.

Ud over at skade spilleren vil ilden blokere for rummets udgange. Spilleren vil altså i et rum med ild kun være i stand til at bevæge sig tilbage til rummet han kom fra.

Spilleren kan fjerne ild for at komme videre igennem spillet. Dette gøres med nogle af de mange objekter der er i stand til at slukke ild. Mere om dette i sektion 1.3.

Hvis et rum indeholder ild med en størrelse på under 3, har det mulighed for at vokse. For hvert femte skridt som spilleren tager (hvert femte rum-skift), vil al ilden i spillet med en størrelse på under 3 øge sin størrelse med 1.

Røg

Nogle af de rum spilleren kommer forbi indeholder røg. Når spilleren bevæger sig ind i et rum med røg, vil han tage skade. Skaden er konstant på 5.

I modsætning til ild, vil røg ikke blokere for et rums udgange.

Røg kan ikke fjernes på nogen måde.

Låste rum

Spilleren kan støde på et låst rum. Spilleren vil ikke være i stand til at bevæge sig ind i et låst rum.

Et låst rum kan låses op med en nøgle, som er en af de mange objekter i spillet.

Items

Brugerne vil igennem spillet møde objekter som kan anvendes. Disse er tegnet i Rum-vinduet som en del af baggrunden. Brugeren er i stand til at få spilleren til at samle objekter op i sit inventar ved at klikke på billedet af objektet.

Spilleren er kun i stand til at bære på ét objekt ad gangen.

Når spilleren opsamler et objekt, vil det vise sig i nederste højre hjørne af spilvinduet under teksten 'Inventory'.

Når man bærer på et objekt, kan man interagere med det på tre måder:

- Anvend: Ved trykke på 'Use'-knappen placeret i kolonnen til højre i spilvinduet, vil man 'anvende' det objekt spilleren bærer. Det specifikke resultat af anvendelsen afhænger af hvilket objekt spilleren bærer på.
- Smid: Ved at trykke på 'Drop'-knappen placeret i kolonnen til højre i spilvinduet, vil spilleren lægge sit opsamlede objekt fra sig igen. Objektet bliver placeret i rummet som spilleren står i på det pågældende tidspunkt, og det vil være muligt at genopsamle det.
- Inspicér: Ved at trykke på 'Inspect'-knappen i kolonnen til højre i spilvinduet, vil det være muligt at få en kort beskrivelse af det objekt spilleren bærer på. Beskrivelsen printes i tekstarealet øverst i spilvinduet.

Fire Escape indeholder mange forskellige objekter. Nogle kan anvendes og andre kan ikke. Herunder er en liste af spillets forskellige slags objekter og hvad de kan anvendes til:

Objekt	Anvendelse
Spand	Ved at anvende spanden, er spilleren i stand til at slukke ild. Spanden bliver dog nødt til at være fyldt op først. Dette opnås ved at anvende spanden i et rum der indeholder vand som f.eks. badeværelset. En fyldt spand er i stand til at sænke størrelsen på ilden med 1. Spanden kan kun anvendes på ild én gang, da den smelter.
Lille brandslukker	Den lille brandslukker kan anvendes på ild for at sænke dens størrelse med 2. Den lille brandslukker kan kun anvendes én gang, da den tømmes.
Stor brandslukker	Den store brandslukker kan anvendes på ild for at sænke dens størrelse med 3.

	Den store brandslukker kan kun anvendes én gang, da den tømmes.
Nøgle	Nøglen kan anvendes til at låse låste rum op. Spilleren bliver nødt til at anvende nøglen i et rum med en udgang der fører til et låst rum for at låse det op.
Yankiebar	Tildeler spilleren fuldt liv ved brug. Slettes efter ét brug.
Dukke	Kan leges med. Gør ellers ikke noget interessant.
NonUseableItem	Spillet er spækket med såkaldte NonUseableitems. Disse kan samles op men ikke anvendes til noget.

Når spilleren anvender objekter, vil de oftest give point. Mængden af point afhænger af hvilket objekt spilleren bærer på og hvordan det bruges. I visse tilfælde kan anvendelsen af objekter give minuspoint hvis det anvendes på en forkert måde.

Afslutning af spillet

Det er to måder spillet kan afsluttes på; Gennemførsel eller død.

Gennemførsel opnås ved at nå til et specifikt rum. I dette tilfælde, gælder det om at nå udenfor, altså til rummet 'Outside'. Så snart spilleren bevæger sig ind i dette rum, skifter billedet i rum-vinduet til gennemførselsbilledet. Når spillet gennemføres, vil alle knapper i spillet deaktiveres, så man ikke kan spille videre. En knap 'OK' viser sig i midten af rum-vinduet. Hvis brugeren klikker på denne, vil man blive ført tilbage til startmenuen.

Død: Spillet afsluttes hvis spilleren dør. Spilleren dør når al hans liv er er opbrugt. Spilleren starter med 100 liv-point. Dette antal sænkes som resultat af spillerens handlinger. F.eks. ved kontakt med ild eller røg. Når spillerens liv-point når nul, vil spillets knapper deaktiveres, så man ikke er i stand til at spille videre. En tekst vil dukke op som spørger om brugeren har lyst til at prøve igen. Brugeren har her mulighed for at trykke på to knapper; 'Yes' eller 'No'. Ved at trykke på 'Yes' vil man blive ført tilbage til startmenuen, hvorfra man er i stand til at starte spillet igen. Hvis brugeren trykker på 'No', vil programmet lukke ned.

Fælles for de to måder af afslutte spillet på, er at brugerens highscore gemmes. Igennem spillet opsamler spilleren en score ved at anvende objekter. Denne score gemmes i systemet, hvis den overstiger den 10. højeste score der fandtes i forvejen. Hvis det lykkes brugeren at gennemføre spillet, vil der pålægges en mænge bonuspoint til spillerens score. Mængde af bonuspoint er baseret på antallet af skridt spilleren har taget. Jo færre skridt, desto flere point.

Grænseflade

Når spillet er i gang, kan brugeren finde hjælpsom information på de forskellige tekstfelter rundt omkring i spillet. Øverst i spilvinduet findes det hoved-tekstfeltet. Her printes beskrivelser af hvad der foregår i spillet. Dette kunne f.eks. være når spilleren skifter rum, tager skade, anvender objekter og meget mere.

Nede i venstre hjørne findes en lille brandmands-kanin. Han har til formål at hjælpe brugeren igennem spillet. Hvis brugeren klikker på 'Help'-knappen i kolonnen til højre i spilvinduet, vil et tekstfelt blive synligt ved siden af brandmanden. Her står en lille hjælpende tekst der kan hjælpe brugeren på vej. Den hjælpende tekst afhænger af hvor lang brugeren er nået igennem spillet.

Meget hjælpsom information kan også findes inde i rum-vinduet af spillet. F.eks. vises der et grønt rektangel henover spillerfiguren. Denne indikerer hvor mange liv-point spilleren har tilbage. I øverste venstre hjørne af rum-vinduet står navnet på det rum som spilleren befinder sig i. Øverst til højre af rum-vinduet angives antallet af skridt spilleren har taget. Akkurat herunder vises den spillerens score indtil videre.

2.2 Highscore

Hvis brugeren vælger at trykke på 'Highscore'-knappen på startmenuen, vil scenen skifte, og brugeren vil blive præsenteret for en liste over de bedste scores opnået. Listen viser 10 eller mindre scores afhængig af antallet.

Efter et spil, vil brugeren være i stand til at se sin egen score, hvis den overstiger en af dem som i forvejen var på listen.

Disse highscores gemmes selvom programmet lukkes ned. Brugeren har altså mulighed for tilgå sine gamle highscores.

Ved at trykke på 'Back'-knappen vil man blive ført tilbage til startmenuen.

2.3 Quit

Lukker programmet ned.

V. Projektlog

1. Logbog

Herunder findes et link til gruppens Github wiki, hvor alt indholdet består af logbøger af projektdagene, frivillige arbejdsdage og vejledermøderne. Af hensyn til censor er gruppens repository gjort offentligt.

Link til logbøger:

https://www.github.com/tlaur18/Semesterprojekt---World-of-Zuul/wiki

VI. Bilag

Bilag 1 - Projektgrundlag

Projektgrundlag

Universitet:	Syddansk Universitet Odense
Fakultet:	Det Tekniske Fakultet
Uddannelse:	Software Engineering
Kursusnavn:	SI1-OOP Semesterprojekt
Afleveringstype:	Projektgrundlag
Projekttitel:	Brandsikkerhedsspil? / Fire Escape?
Projektgruppe:	Gruppe 02

Projektforfattere:

Jacob Wowk Jørgensen	*jacjo18@student.sdu.dk*
Thomas Steenfeldt Laursen	*tlaur18@student.sdu.dk*
Morten Jensen	*mortj18@student.sdu.dk*
Mathias Lilleskov	*mathj18@student.sdu.dk*
Alexander Nguyen	*alngu18@student.sdu.dk*

Projektvejledere:

Lone Borgersen	*lobo@mmmi.sdu.dk*

Dato

12. oktober 2018

Vi er igennem vores brainstorm fundet frem til, at vi vil beskæftige os med følgende grundlæggende problem:

For mange mennesker er ofre i brandulykker

Introduktion

Dokumentation af problemet

Det er nemt at se, at dette problem viser sig i virkelighedens verden. Hvert år forekommer brandulykker som forvolder skader for flere millioner af kroner og i de værste tilfælde skader eller endda dræber menneskerne indblandet. Ifølge beredskabsstyrelsen er der i de seneste ti år omkommet 70 mennesker fordelt på 68 brande i gennemsnit pr. år i Danmark[10]. Ydermere, er antallet af omkomne i brand steget med hele 20 % i Danmark fra år 2016 til 2017[11]. Problemet er altså aktuelt i nutidens samfund, og det er selvfølgelig ikke optimalt.

Problemets historie og baggrund

Problemet er ikke noget som er opstået for nyligt. Dødelige brande har raseret helt siden tidernes morgen, og det vil de højest sandsynligt blive ved med. Dog er der flere faktorer som er med til at gøre dette problem aktuelt. Ifølge en undersøgelse foretaget af ingeniørforeningen IDA, taler for få forældre med deres børn om brandsikkerhed[12]. Kun 42 procent af de adspurgte havde snakket med deres familie/børn om hvordan man skulle forholde sig i tilfældet af brand. At langt fra alle har informeret deres børn om hvad de skal gøre hvis en brand bryder ud, vil resultere i, at børn ikke er i stand til at redde dem selv, hvilket kan føre til deres død. Et bud på hvorfor forældrene ikke tager emnet om brandsikkerhed op med sine børn, er fordi de ikke selv har den nødvendige viden[13]. Hvis flere forældre altså selv vidste hvordan man skulle håndtere en brandsituation, ville de med større sandsynlighed videreføre denne viden.

Derudover er det ifølge undersøgelsen kun 31 procent af danskere der har anskaffet sig ordentligt brandbekæmpelsesudstyr i form af brandslukkere og -tæpper osv. Dette vil selvfølgelig gøre det noget sværere at bekæmpe en allerede udbrudt brand.

Det viser sig, at de abiotiske faktorer også har noget at sige. Klimaforandringer i form af global opvarmning vil gøre naturen mere tør og tilbøjelig til at antænde[14]. Klimaforandringens påvirkning af naturen har især vist sig i denne sommer, hvor solen har været fremme mere end nogensinde før. Mellem den 1. maj og 10. juli rykkede det danske beredskab ud 1021 brandsituationer[15]. Dette er en stigning på 150 procent siden sidste år. Disse brande vil selvfølgelig være med til at øge faren for at komme til skade, men de vil endda også forårsage at tørkeperioder som disse forekommer hyppigere[16]. Naturbrande udleder nemlig CO_2 i atmosfæren, hvilket har været årsagen til at Klimaforandringerne startede. Dog udledes der en meget lille mængde CO_2, så påvirkning af klimaet er meget begrænset.

Interesseparter

Den almene person vil have en vis mængde interesse inden for brandsikkerhed, dog vil specielt familier med børn have en ekstra interesse i at deres børn ved hvordan man agerer i brand.
Mange familier får ikke snakket med deres børn omkring flugtveje, og hvordan man agerer i tilfælde af en brand.

[10] (Beredskabsstyrelsen, 2018)
[11] (Beredskabsstyrelsen, 2018)
[12] (IDA, 2017)
[13] (Danske Beredskaber, 2017)
[14] (Sander, 2018)
[15] (Sander, 2018)
[16] (Jensen, 2018)

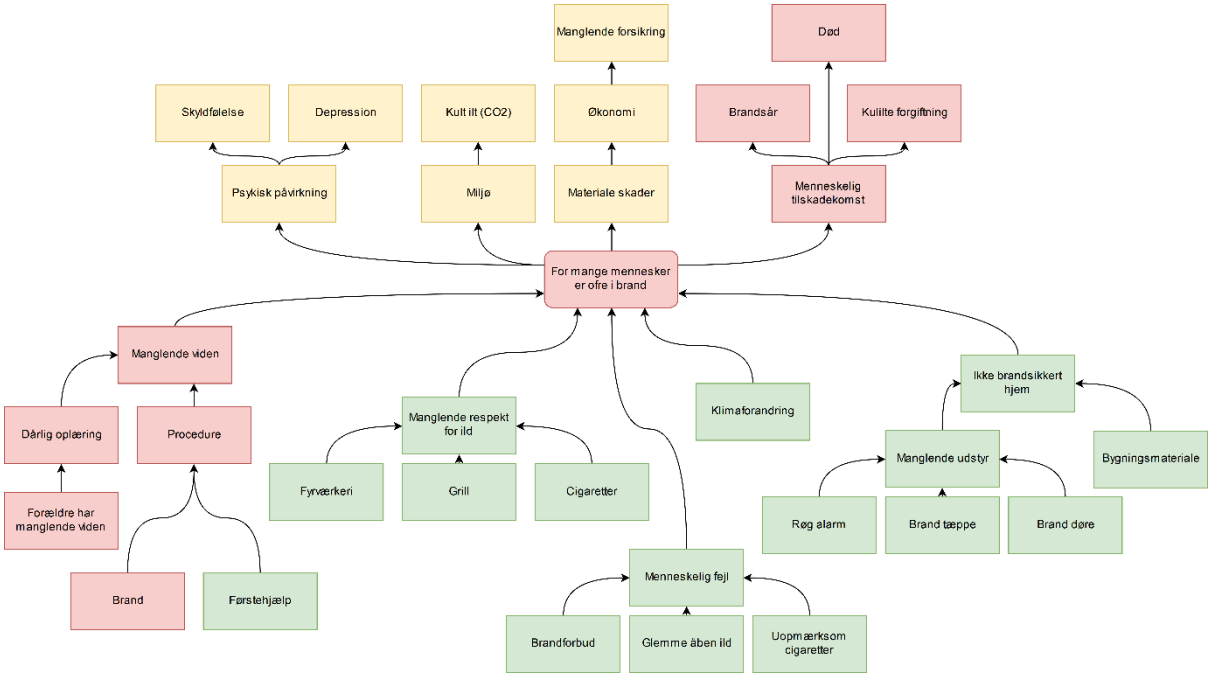
Derudover vil skoler såvel som andre undervisningsinstitutioner have interesse for at deres unge mennesker ved hvordan man skal forholde sig i den slags situationer, også her vil spillet have en interesse af de samme grunde som hos børnefamilierne.

Som den sidste interessepart vi vil nævne, er beredskaberne, som har interesse i at så stor del af befolkningen som muligt, kender til hvordan man skal forholde sig i tilfælde af brand eller ulykke.
Deres interesse ligger på i effektiviteten af udførslen af deres arbejde, men også i når de selv underviser på skoler og institutioner, f.eks. er de fleste beredskaber med i Beredskabsstyrelsens uge 40 kampagne, der handler om at sætte fokus på brandsikkerhed i de almene hjem[17].

Her kan problemet være at eventuelle oplysninger kan blive glemt på vej hjem, så der derfor bør findes en oplysningsmetode der sikrer at informationen bliver givet til hjemmet og det er noget som hjemmene er opmærksomme på.

Problemets sammenhæng og kompleksitet

Da vi valgte at vores overordnede problem skulle omhandle brandsikkerheden pga. det samfundsmæssige aspekt der i at informere befolkningen. I den forbindelse har vi udarbejdet et problemtræ, som skal hjælpe os med at strukturere problemet og sammenhængen mellem problemerne, årsager og konsekvenser. Efterfølgende afgrænsede vi problemtræet, for at komme nærmere vores problemstilling og konkretisere. I vores problemtræ, som er indsat nedenfor, har vi markeret de afgrænsede problemer med rødt. Problemtræet kan ses på figuren herunder. En lidt større version af billedet kan ses i bilag 1.



Figur 7 - Problemtræet. De rød-farvede kasser er dem vi har valgt at fokusere på i dette projekt

Projektrammerne

I forbindelse med undervisningen i Objektorienteret Programmering, skal vi udarbejde dette projekt. Der skal igennem forløbet udvikles og dokumenteres et program ved hjælp af objektorienteret programmering. Vores program skal bygges ud fra den udleverede kildekode til World of Zuul.

[17] (Beredskabsstyrelsen, 2017)

World of Zuul er et tekstbaseret spil, hvor brugeren interagerer med en konsolbaseret brugergrænseflade. Igennem forskellige kommandoer kan brugeren bevæge sig mellem forskellige rum.

Vi skal altså udvikle World of Zuul-kildekoden, så vi får skabt et program, som passer til vores specifikke problemstilling, men stadig følger World of Zuul-konceptet med bevægelse igennem rum. Det er blandt andre krav, at vi skal tilføje flere rum, flere kommandoer, tekstbaseret brugergrænseflade (1. delforløb), grafisk brugergrænseflade (2. delforløb) osv. Disse rammer for projektet tilføjer både nogle forskellige begrænsninger og muligheder. I første delforløb, hvor vi skal bruge en tekstbaseret brugergrænseflade, bliver vi begrænset i vores evne til at beskrive sceneriet for brugeren. Dette vil blive noget nemmere i 2. delforløb med den grafiske brugergrænseflade. Derudover er vi selvfølgelig i projektet også begrænset til at anvende objektorienteret programmering til udviklingen af World of Zuul. Objektorienteret programmering tilføjer dog på samme tid en stor mængde muligheder. At programmet skal være objektorienteret, gør det muligt at lave klasser og objekter som går igen igennem hele programmet. Man kan altså genbruge store dele af koden og på den måde gøre programmet nemmere både at udvikle og vedligeholde.

Problemformulering

Brande har vist sig at være et stort problem i Danmark med ca. 70 branddrab og endnu flere kroner i skade om året. Vi har set flere årsager til dette problems eksistens. F.eks. klimaforandringernes indtørring af naturen. Mere relevant er nok den manglende viden hos befolkningen. Især nutidens børn bliver ikke tilstrækkeligt udlært i hvordan de skal handle i tilfældet af brand. Dette er til dels forældrenes skyld, som nemlig heller ikke besidder den tilstrækkelige viden om brandsikkerhed. Dette problem er til gene for en stor del af befolkningen som f.eks. børnefamilier som er i fare for at problemet gør dem ude af stand til at handle tilstrækkeligt, hvis de selv skulle befinde dem i ved en brand.

I projektet skal vi videreudvikle det simple tekstbaserede program World of Zuul for at til et stykke software som skal kunne bidrage til løsningen af vores grundlæggende problem. Vi undrer os derfor om det er muligt ved hjælp af et stykke software løse dette problem. Vi ønsker dermed ved dette projekt at finde ud af om dette kan lade sig gøre, og hvordan softwaren i så fald skulle løse problemet.

Vores primære fokus på problemet vil blive den manglende viden om brandsikkerhed blandt Danmarks indbyggere - både voksne og børn. Vi ønsker at skabe et program de som ved hjælp af en kombination af underholdning og læring er i stand til at undervise børn og unge i hvordan man skal opføre sig hvis det brænder. Hovedspørgsmålet, som vi ønsker at besvære igennem dette projekt, er altså:

> Kan vi udvikle et læringsspil til børn og unge som kan bidrage til deres forståelse for hvordan brandsituationer skal håndteres, og som samtidig opfordrer til at der tages en snak om emnet i hjemmet?

For at kunne besvare projektet hovedspørgsmål, bliver vi nødt til at løse og besvare følgende opgaver og spørgsmål:

- Kan vi bygge et underholdende og læringsrigt program ud fra en World of Zuul-skabelon?
- Er det muligt at bygge et program som udspiller et scenarie hvor man befinder sig i en brændende bygning og skal finde ud?
- Kan man i programmet bygge en brændende bygning, så den implementerer rum-opdelingen fra World of Zuul?
- Kan man lave et børnevenligt spil i forhold til et meget seriøst og skræmmende emne som brandsikkerhed?

- Kan en form for konkurrence-element i form af point og/eller andet øge interessen for programmet?
- Skal programmet udlodde en gevinst når man har gennemført spillet for at øge motivationen og opfordre til en dialog i hjemmet?
- Kan man igennem spillet motivere en samtale om brandsikkerhedsemnet i hjemmet gennem direkte beskeder til brugeren?

Opgaverne der skal løses er:

- Få dannet os en forståelse for kildekoden.
- Få dannet nogle specifikke krav til programmet - både funktionelle og ikke-funktionelle.
- Få dannet et overblik over programmets designbindinger og -valg.
- Implementere flere funktionaliteter såsom forhindringer, genstande der kan samles op og derefter anvendes, personer/ting der kan interagere med brugeren (NPC).
- Lave løbende tests på koden.

Metoder

Vores metoder bygger på et praktisk problem, da vores løsningsmetode bygger på en konstruktion, dog har vi brugt mange af de teoretiske problemstillinger for at tilgå den viden der er brugt her i problemformuleringen for at undersøge omfanget af vores problem.

Når vi skaber et program, har vi fire forskellige elementer; krav, design, kode implementering og test. Disse er alle ting man bør overveje hvis man skaber en konstruktion. Vi har først krav, som er vores funktionelle og ikke-funktionelle krav. Det er dem der sætter vores mål med programmet. Bliver disse overholdt har vi et velfungerende program. Det er derefter vigtigt at tænke på design som er vores løsningsforslag til kravene, det er dem der bestemmer hvordan vores produkt skal udformes visuelt, arkitektonisk og mere specifikke krav så som programmeringssprog. Man skelner mellem designbindinger og -valg, som henholdsvis er noget som kunden og programmøren beslutter. Derefter kommer selve implementeringen af koden, og her er mange basale ting som skal overvejes, når vi koder programmet. Det er ting som valg af navne af klasser så man ikke bruge allerede eksisterende navne, teste sine funktioner inden man implementere dem og mange andre ting. Derudover er det vigtigt at teste sit program. Det testes blandt andet om alle vores krav er overholdt, om kvaliteten af spillet lever op til vores forventninger og om ens kode overhovedet fungerer.

I den teoretiske del har vi brugt søgningsværktøjer til at finde emnerelevante sider der kunne hjælpe os med at opnå bedre indsigt i omfanget af problemet, og mere specifikt kunne bygge denne viden oven i vores projekt. Derudover har vi også taget kontakt til et firma, i vores tilfælde et brandvæsen, som beskæftiger sig med forebyggende arbejder i forhold til brandsikkerhed, for at få deres syn på hvor stort omfanget af vores problem er, og hvordan de tænker vores program kan være med til at afhjælpe problemet. I vores tilfælde er vores løsningsmetode et program som igennem vores problemstillinger giver brugeren noget viden omkring emnet, og deraf kan være en vej til at hjælpe problemet.

Tidsplan

Opgave\Uge	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
Samarbejdsaftale	X	X				Efterårsferie										Juleferie
Vejledningsaftale	X	X														
Idégenerering		X	X	X	X											
Problemanalyse			X	X	X											
Problemformulering			X	X	X											
Postersession					X											
Aflevering af Projektgrundlag					X											
Første udkast							X	X	X	X						
Midtvejsseminar										X						
Andet udkast											X	X	X	X	X	

Bilag 2 - Samarbejdsaftale

Forventninger til og mål med projektet

- Det er et fælles mål blandt gruppemedlemmerne at blive bedre til fagets indhold. Alle ønsker, at alle får en forståelse for det hele, så vi fælles niveau i afslutningen af projektet.
- Gruppemedlemmerne bestræber så vidt muligt at fremstille et projekt som alle er tilfredse med.
- Gruppens medlemmer skal så vidt muligt gøre deres bedste. Dog er der en grænse for hvad hver især kan præstere, og man kan selvfølgelig ikke kræve mere. Derfor er det ikke nødvendigt at score topkarakterer, bare gruppemedlemmerne gør sit bedste. Selvom topkarakterer selvfølgelig er hvad vi alle håber på.
- Det forventes, at alle medlemmer af gruppen leverer det bedst mulige materiale, når personen har fået en opgave. Herunder forventes det, at personen selv har læst korrektur på sit materiale.

Hvad motiverer den enkelte i gruppen til at gøre en indsats?

- Ansvarsfølelsen over for de andre i gruppen gør at man yder en bedre præstation.
- Den afsluttende karakter

Arbejdstider

- Møder klokken 8:30 hver tirsdag på grund af dumme bustider. Arbejder til klokken 16 medmindre gruppens medlemmer er enige om at tage tidligt fri.
- Møder klokken 8:15 hver fredag medmindre der er dumme bustider. Vi arbejder selvfølgelig til forelæsningen begynder klokken 12.
- Hvis der gives hjemmearbejde, forventes det, at man afsætter tid derhjemme til at lave det.

Gruppemøder

- Tirsdag og fredag mødes vi. Eventuelt kan vi arrangere møder uden for skoletiden og arbejde hvis det bliver nødvendigt.
- Hvis man kommer for sent: Hvis man ikke overholder de planlagte mødetider, skal der pålægges en form for straf medmindre årsagen kan begrundes godt. Dette kan være i form af kage eller bajer. Hvis man ved, at man kommer for sent, skal man skrive en besked til gruppen.
- Hvis et gruppemedlem ikke møder op til et planlagt møde, skal man skrive en besked til gruppen før mødet starter. Her kan der pålægges en eventuel straf, hvis årsagen ikke er begrundet tilstrækkeligt. Hvis et gruppemedlem gentager et sådant fravær flere gange, tages der en snak med gruppemedlemmet for at finde en løsning. Hvis det ikke er muligt at komme til enighed, må vejlederen involveres. Det samme gælder hvis man ikke laver det aftalte arbejde.
- Det bestræbes at minimere overflødige diskussioner, men de tillades stadig i små mængder for at gøre arbejdsdagene udholdelige.

Organisering af møder - ordstyrer - referent – logbog?

- Til hvert vejledermøde får ét gruppemedlem rollen som ordstyrer og en anden får rollen som referent. Rollerne som ordstyrer og referent følger det forudfremstillede skema.
- X'et under "Referent" og "Ordstyrer" indikerer hvem der får tildelt disse roller. Efter hver gang rykkes de to x'er én plads ned, så rollerne går videre:

	Referent	Ordstyrer
Alexander Nguyen		X
Jacob Wowk	X	
Mathias Lilleskov		

Morten Jensen		
Thomas Steenfeldt		

- Logbog skrives efter hvert eneste møde - både uden og inden for skoletiden. Rollen som lobogs-skriver følger det forudfremstillede skema:

	Logbog
Alexander Nguyen	X
Jacob Wowk	
Mathias Lilleskov	
Morten Jensen	
Thomas Steenfeldt	

- Det forventes, at alle de, ifølge logbogen, aftalte deadlines overholdes. Er dette ikke muligt, skal det angives i logbogen hvorfor tidsgrænsen ikke er overholdt.

Fælles versus individuelt arbejde - hvor meget? – hvornår?

- Hver gang vi mødes på skolen, sætter vi os i samme lokale og arbejder fælles. Dette kan dog foregå på forskellige måder. Nogle gange kan vi arbejde sammen om en ting, og andre gange kan arbejdet deles op. Dog skal vi altid kunne have mulighed for kunne spørge hinanden til råds.
- Individuelt arbejde aftales fra gang til gang og laves hjemme.

Arbejdsdeling i forhold til skriftligt arbejde?

- De forskellige afsnit i rapporten fordeles blandt gruppens medlemmer for at gøre det skriftlige arbejde effektivt. Hvilke afsnit der skal fordeles til hvem tages løbende igennem processen.

Vejledermøder - hvor tit? - Forberedelse?

- Der afholdes ét vejledermøde hver uge for at sikre at vi altid er på rette spor.
- Flere / færre vejledermøder planlægges efter behov.
- Gruppen har pligt til at aflevere materiale via e-mail til vejlederen senest kl. 12 dagen før materialet skal behandles.

Kursusdeltagelse og opgaveløsning - individuelt eller fælles?

- Med mindre andet aftales, deltager alle gruppens medlemmer til alle forelæsninger.
- Så vidt muligt skrives kommentarer i koden for at gøre den så forståelig som mulig.

Deadlines

- Uge 41: Postersession med præsentation af projektgrundlag
- Uge 46: Midtvejsseminar (Udkast til rapport og foreløbig kode)
- Uge 50 (14. december): Aflevering af kode
- Uge 51 (21. december): Aflevering af rapport
- Uge 1 - 4: EKSAMEN!!! ÅH NEJ DOG!

Kontaktoplysninger:

- Alexander Nguyen
  - Mail: alngu18@student.sdu.dk
  - Telefon: 53344014
- Jacob Wowk
  - Mail: jacjo18@student.sdu.dk
  - Telefon: 21626779

- Mathias Lilleskov
    - Mail: mathj18@student.sdu.dk
    - Telefon: 20688919
- Morten Jensen
    - Mail: mortj18@student.sdu.dk
    - Telefon: 22300758
- Thomas Steenfeldt
    - Mail: tlaur18@student.sdu.dk
    - Telefon: 30953669

Bilag 3 - Vejledningsaftale

Forventning til vejleder

- Procesorienterede projektvejleder som udgangspunkt, hvor vejlederen ikke kommer med direkte løsninger, men sørger for at gruppens arbejdsproces forløber korrekt. Dog må der gerne være dele af andre vejledningstyper som produktvejledning, hvor vejlederen kommer med forslag til anvendelse og tolkning af teori og metode.
- At vi kan bruge vores vejleder til at modtage kritik på udleverede dokumenter og finde brugbart materiale.
- Det forventes at vejlederen møder til vejledermøderne til tiden og har forberedt ved at læse eventuelle dokumenter.

Mødelokation og mødetid

- Gruppen og vejleder mødes om tirsdagen i lokale ø27-507a-2 til afholdelse af vejledermøde.
- Vi vil stræbe efter at udsende indkaldelsen senest klokken 12 dagen før, hvor vi også vedhæfter de dokumenter som vores vejleder måtte skulle forberede sig på op til mødet.
- Mødet starter klokken 13.30 - 14.15, men det er ikke ensbetydende tiden skal overholdes.
- Vi starter mødet med en statusopdatering hvor vi snakker om hvad der er foretaget siden sidst. Derefter følges der op med en gennemgang af den pågældende dagsorden. Til sidst i mødet snakkes der om hvad det næste skridt i projektet er og hvad der skal forberedes til næste vejledermøde.
- Det forventes at alle gruppens medlemmer er til stede til vejledermødet.
- Det forventes at alle er velforberedte til vejleder mødet dvs.. at alle medlemmer har læst og løst opgaver til E-tiviteter, samt har gennemgået dagsordenen for dagen.
- Referent sender referatet til vejlederen samme dag som vejledermødet er afholdt. Referent vil få rollen som ordstyrer ved næste vejledermøde og har ansvaret for at indkalde vejlederen

Bilag 4 - Rapportkontrolskema

<table>
<tr><th colspan="3">A. Produktrapport</th></tr>
<tr><th>Kapitel</th><th>Krav</th><th>Opfyldt +/-</th></tr>
<tr><td>Omslag</td><td>Indeholder omslaget projekttitel, uddannelsesinstitution, fakultet, institut, uddannelse, semester, kursuskode, projektperiode, vejleder, projektgruppe og projektdeltagere (fornavn, efternavn, sdu-email)?</td><td>+</td></tr>
<tr><td>Titelblad</td><td>(Som omslag ekskl. evt. illustration + evt. kildehenvisning til evt. omslagsillustration. Omslaget kan udgøre både omslag og titelblad. Hvis der medtages selvstændigt titelblad, så er titelbladet rapportens første højre side)</td><td>+</td></tr>
<tr><td>Resumé</td><td>Omfatter resuméet:
Den behandlede problemstilling - hvad blev der arbejdet med og hvorfor?
Fremgangsmåden - anvendte metoder - hvordan blev der arbejdet med det? (hvordan angreb I problemet og hvordan realiserede I løsningen (hvem, hvad, hvornår og hvorfor)
Hovedresultater og konklusioner – hvad kom der ud af arbejdet?
(max 1 side)</td><td>+</td></tr>
<tr><td>Forord</td><td>Indeholder forord hensigten med rapporten, målgruppe, forhistorie, anerkendelser, afleveringsdato samt underskrifter af alle projektdeltagere?
Bemærk: Projektdeltagernes aktive deltagelse i projektforløbet anerkendes gensidigt ved projektdeltagernes underskrifter i rapporten.</td><td>+</td></tr>
<tr><td>Indholdsfortegnelse</td><td>Er der en samlet indholdsfortegnelse for hele projektrapporten?. (Højst to eller tre niveauer i indholdsfortegnelse)</td><td></td></tr>
<tr><td>Læsevejledning</td><td>Er der en vejledning i, hvordan rapporten kan læses, eksempelvis i form af hvilken rækkefølge afsnittene kan læses i og hvordan sammenhængen er mellem de forskellige dele af rapporten, fx mellem produktrapport og bilag?</td><td>+</td></tr>
<tr><td></td><td>Er rapportens målgruppe beskrevet?</td><td>+</td></tr>
<tr><td>Redaktionelt</td><td>Beskriver redaktionelt skriveprocessen og ansvarsområder i skriveprocessen?

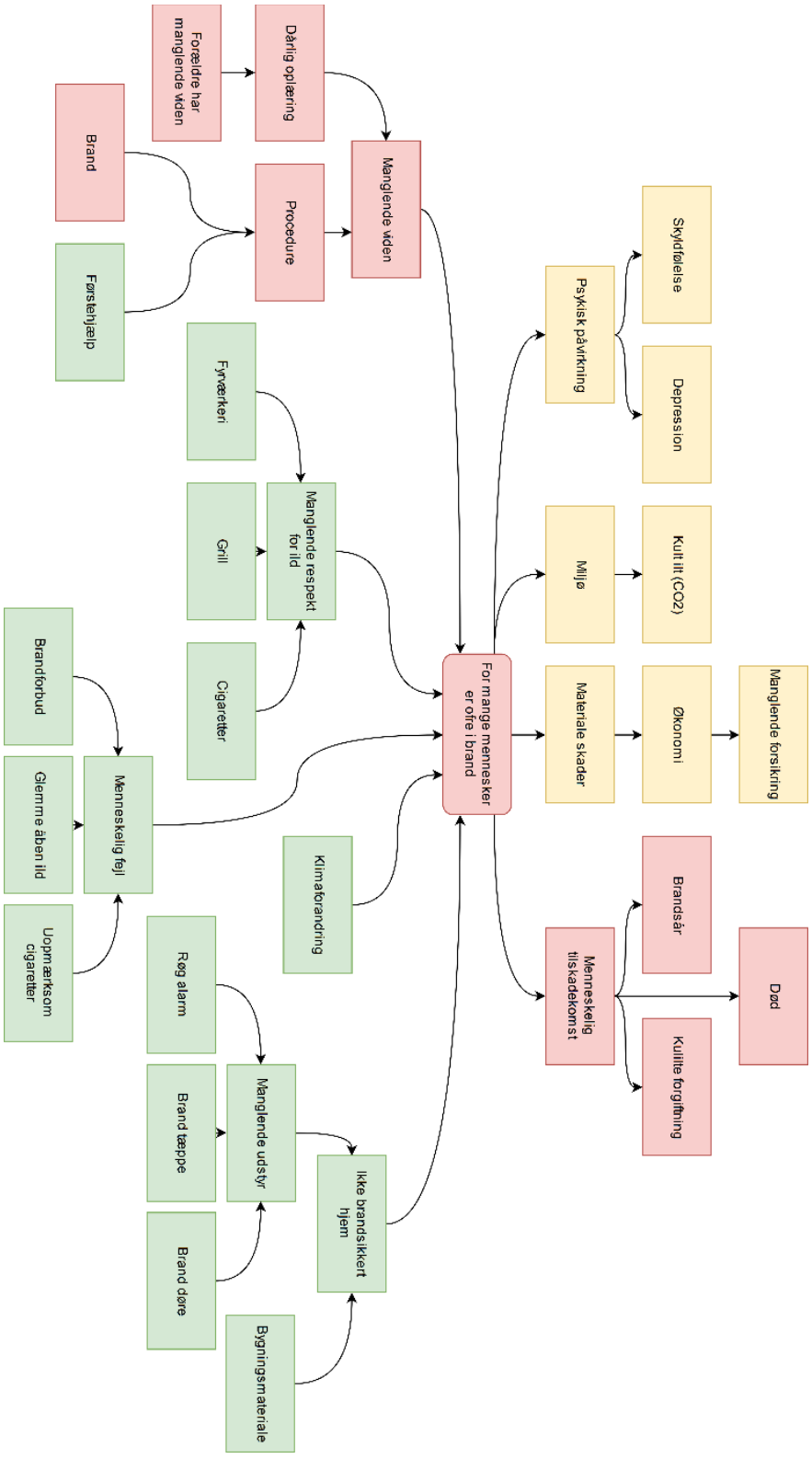
Ansvarsområder kan fx beskrives på fx følgende form:
<table>
<tr><th>Afsnit</th><th>Ansvarlig</th><th>Bidrag fra</th><th>Kontrolleret af</th></tr>
<tr><td>Afsnit a</td><td>Person a</td><td>Person b</td><td>Person a, b, c</td></tr>
<tr><td>Afsnit b</td><td>Person b</td><td>Person a</td><td>Person a, b, c</td></tr>
</table></td><td>+</td></tr>
</table>

	Afsnit c / Person c / Person b / Person a, b, c	
Ordliste	Er der en kort beskrivelse af de fagtermer der bruges gennem rapporten?	+
Indledning	indeholder indledningen problemanalysen, herunder en redegørelse for projektets rammer og en analyse af det udleverede framework? Jfr. projektgrundlaget	+
	Indeholder indledningen problemformuleringen? jfr. projektgrundlaget.	+
	Indeholder indledningen en beskrivelse af metoder og tidsplan? jfr. projektgrundlaget.	+
Hovedtekst	Hvis der indgår et teoretisk problem: Omfatter hovedteksten et teori-afsnit?	+
	Omfatter hovedteksten krav formuleret af jer, design, brugergrænseflade, centrale dele af programkoden og kendte fejl?	+
	Er der medtaget resultater både fra iteration #1 og fra iteration #2?	+
Diskussion	Omfatter diskussionen hvad der er opnået, og hvad der ikke er opnået i projektet i forhold til det forventede som beskrevet i indledningen. Hvad er styrkerne og svaghederne ved jeres resultater? Kunne I have opnået bedre resultater?	+
Konklusion	Opsummerer konklusionen resultaterne og diskussionen af dem og giver det på problemformuleringen?	+
Perspektivering	Fremtidigt arbejde: Hvad ville de næste skridt i projektet være, hvis der var mere tid? Refleksion: Hvordan ville I gribe projektet an, hvis I skulle starte forfra?	+
Litteraturliste	Er litteratur angivet på en anerkendt form? Er alle former for litteratur som bøger, artikler og hjemmesider medtaget? Er der kildehenvisninger i teksten? Materiale som gruppen ikke selv har fremstillet i dette projekt skal være angivet med kilde! Er alle kildehenvisninger i teksten anført på samme måde? Er der kildeangivelser på figurer, grafer etc. som projektgruppen ikke selv har frembragt?	+
	B. Procesrapport	
Kapitel	**Krav**	**Opfyldt +/-**
Læring og refleksion	Er der en redegørelse for læring og refleksion?	+
Projektarbejdet	Er der en redegørelse for projektarbejdet	+
Identifikationg, analyse og bearbejdning af problemer	Er der en redegørelse for identifikation, analyse og bearbejdning af problemer	(+)
Formidling og kommunikation	Er der en redegørelse for formidling og kommunikation	+
Samarbejde i gruppen	Er der en redegørelse for samarbejde i gruppen	+
Samarbejde med vejleder	Er der en redegørelse for samarbejde med vejleder	+
	C. Kildekode	

Kapitel	**Krav**	**Opfyldt +/-**
Oversigt over projektets kildekode	Er der en oversigt over projektets kildekode, fx filstruktur eller javadoc?	+
	D. Brugervejledning	
Kapitel	**Krav**	
Brugervejledning	Er der en kortfattet brugervejledning?	+
	E. Projektlog	
Kapitel	**Krav**	
Projektlog	Er der en adresse på og et link til projektloggen i Github	+
	F. Interne bilag	
Kapitel	**Krav**	**Opfyldt +/-**
Projektgrundlag	Er projektgrundlaget medtaget?	+
Rapportkontrolskema	Er der et udfyldt rapportkontrolskemaet	+
Andet	Er der andre relevante interne bilag, dvs. materialer produceret af gruppen selv?	+
	G. Eksterne bilag	
Kapitel	**Krav**	**Opfyldt +/-**
Eksterne bilag	Er der medtaget relevante eksterne bilag, dvs. materialer som gruppen ikke selv har produceret med som er nødvendige for at kunne læse rapporten?	

Rapporttekniske elementer		**Opfyldt +/-**
Layout	Er der anvendt samme layout i alle kapitler? Er layout overskueligt/harmonisk?	+
Sprog	Er rapporten skrevet i en neutral sprogtone? Er sproget let læseligt og flydende? Er der udført stavekontrol og kontrol af tegnsætning?	(+)
Sidenummerering	Er der korrekt og konsistent sidenummerering i rapporten?	+
Figurer/diagrammer	Er alle figurer konsekvent nummererede? Er der figurtitel og figurtekst til alle figurer? Er figurtitler og figurtekster dækkende og afklarende? Er figurerne tydelige og læsbare? Er figurerne informationsgivende og i den rette sammenhæng?	+
Tabeller	Er alle tabeller konsekvent nummererede? Er der en forklarende tabeltekst til alle tabeller? Er alle søjler og rækker forsynet med parametre? Er der enheder på alle relevante rækker og søjler?	-
Sporbarhed af begreber	Er der en konsekvent brug af samme betegnelse for et givet begreb igennem rapporten?	(+)

Bilag 5 – Problemtræet



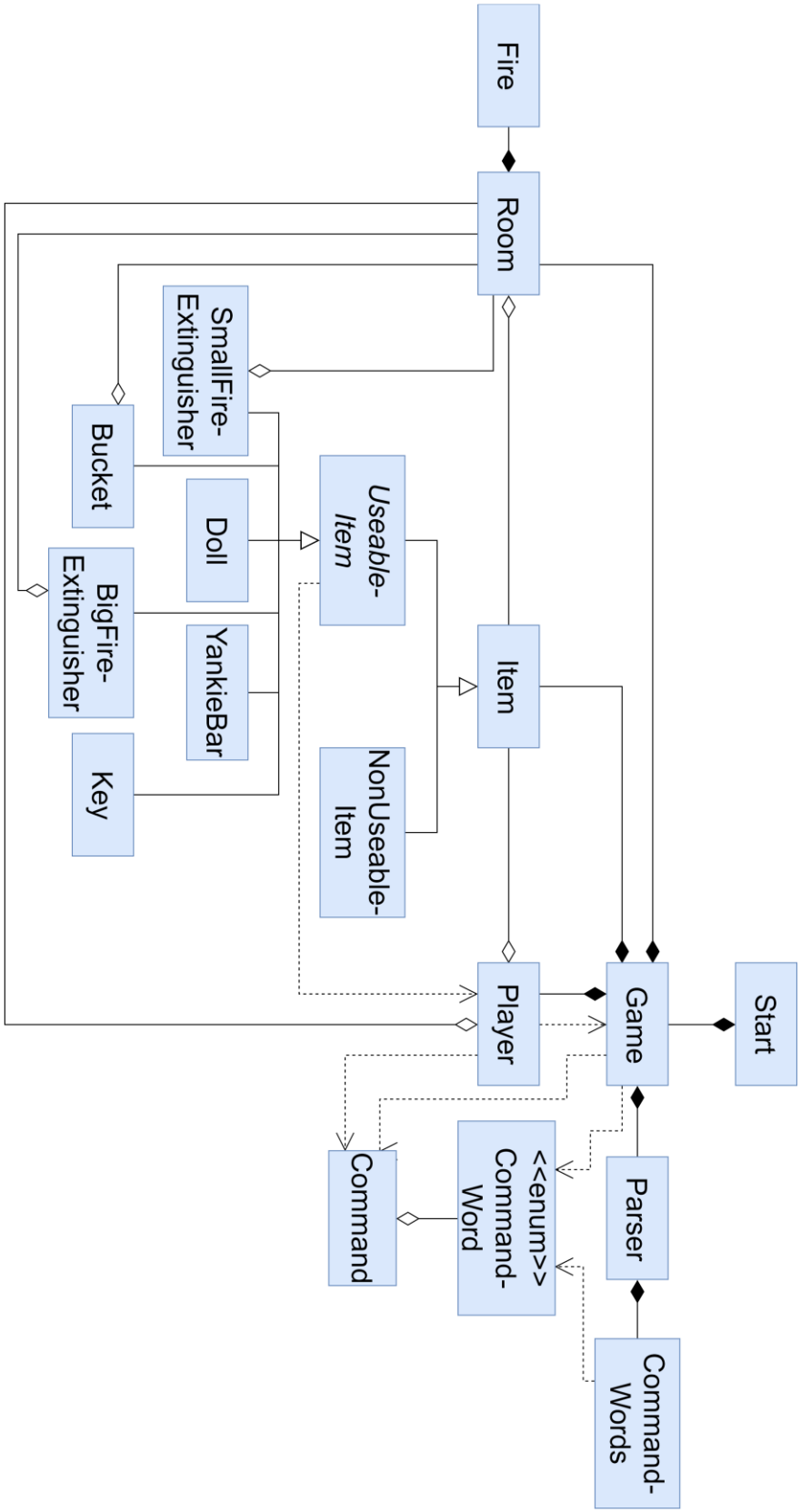
Figur 1

Bilag 6 – Planlagte tidsplan

opgave/uge	Opstart				Fri	1. Iteration				2. Iteration					Fri
	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
samarbejdaftale	x	x														
vejlederaftale	x	x														
Idégenerering		x	x			E	x				x					J
problemanalyse		x	x	x		F										U
problemformulering			x	x		T										L
postersession					x	E										E
Projekt grundlag			x	x	x	R										F
gennemgang af World Of Zuul					x	Å	x									E
Kravspecifikation						R	x				x					R
konceptudvikling						S	x				x					I
Programmering						F		x	x	x	x	x	x	x		E
uml						E		x	x		x	x				
midtvejsseminar						R				x						
Rapportudkast						I			x	x						
processforbedringer						E					x					
Rapportskrivning												x	x	x	x	
Aflevering af spillet														x		

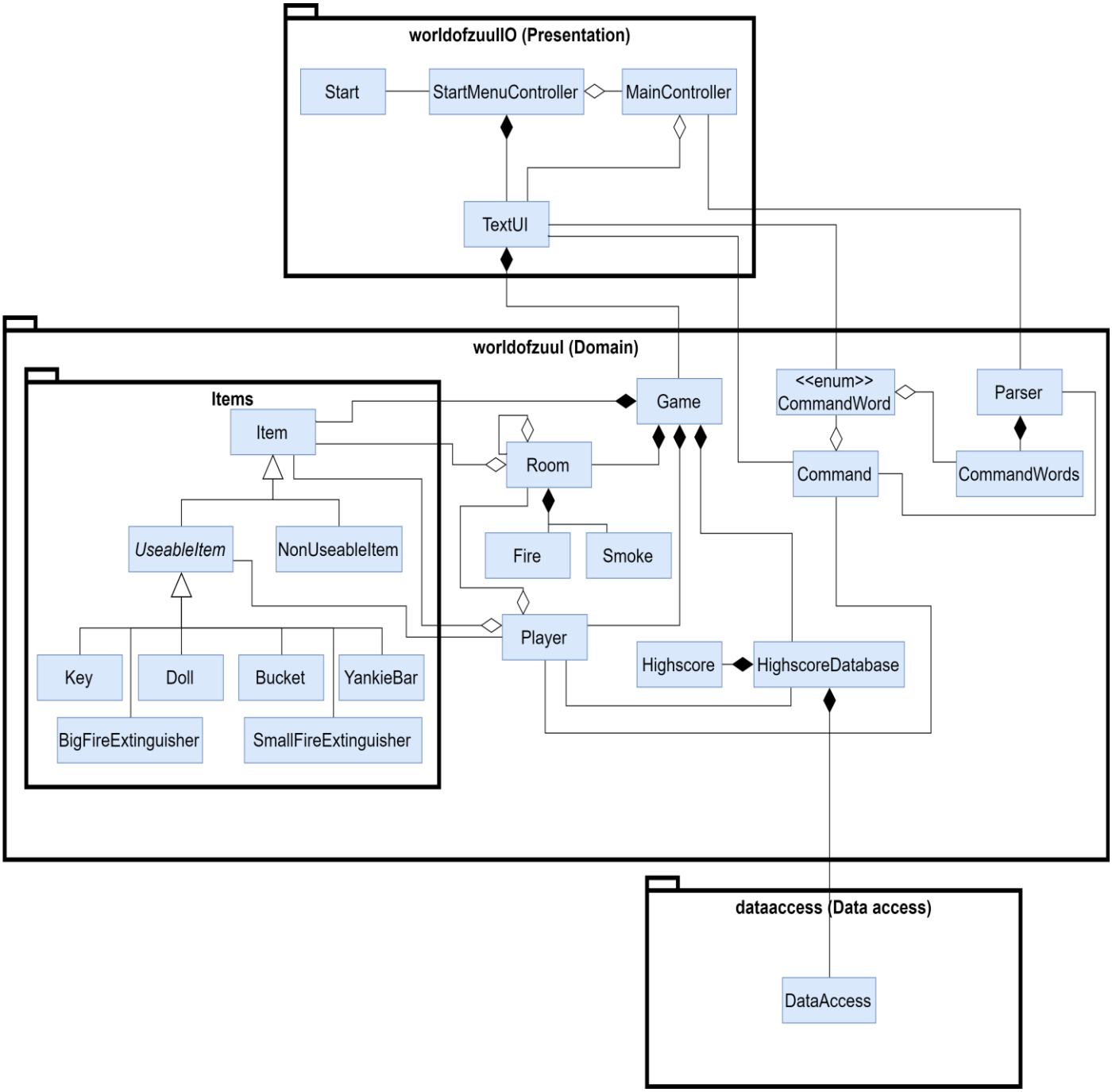
Figur 2

Bilag 7 – Klassediagram



Figur 4

Bilag 8 – Lagdelt arkitektur



Figur 6

Bilag 9 – Den udførte tidsplan

opgave/uge	Opstart					Fri	1. Iteration					2. Iteration					Fri
	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	
samarbejdaftale	x	x															
vejlederaftale	x	x															
Idégenerering		x	x	x	x	E	x	x			x					J	
problemanalyse			x	x	x	F										U	
problemformulering			x	x	x	T										L	
postersession					x	E										E	
Projekt grundlag			x	x	x	R										F	
gennemgang af World Of Zuul						Å	x									E	
Kravspecifikation						R	x	x			x					R	
konceptudvikling						S	x	x			x					I	
Programmering						F		x	x	x	x	x	x	x		E	
uml						E			x	x			x	x	x		
midtvejsseminar						R				x			x	x	x		
1 iterations udkast						I				x							
processforbedringer						E					x						
Rapportskrivning														x	x		
Aflevering af spillet														x			

Figur 30