Sammendrag

Tittel:	VCIIGE		Nr. :
	Visualization of Critical Infrastructure		Dato: 20.05.09
	Interdependencies using Game Engines		
Deltakere:	Robin Skilbrei Sveen		
	Nils Peter Lunden		
	Hans Jacob Koren Lund		
Veileder(e):	Stephen D. Wolthusen		
Oppdragsgiver:	Høgskolen i Gjøvik		
	Nils Kalstad Svendsen og Stephen D. Wolthusen		
Kontaktperson:	Nils Kalstad Svendsen og Stephen D. Wolthusen		
Stikkord (4 stk)	Kritiske infrastrukturer, Spillmotor, Java, Open source		
Antall sider: 100	Antall bilag: 6	Tilgjengelighet(åpen/konfidensiell): åpen	

Kort beskrivelse av bacheloroppgaven:
Utvikling av et verktøy som muliggjør visualisering av kritiske hendelser som følge av gjensidig avhengighet mellom kritiske infrastrukturer. Disse hendelsene omfatter brann med røyk, oversvømmelse og gasseksplosjon. Dette er utviklet ved hjelp av programmeringsverktøy og spillmotor basert på åpen kildekode.

Forord

Denne rapporten er utarbeidet av tre avgangsstudenter ved ingeniørfaglinjen på Høgskolen i Gjøvik som en del av bachelorprosjektet. Gruppens medlemmer har vært Robin Skilbrei Sveen, Nils Peter Lunden og Hans Jacob Koren Lund. Vi valgte denne oppgaven fordi vi fikk inntrykk av at denne ville bli interessant, spennende og lærerik.

Oppgaven viste seg etter hvert å være en god del mer komplisert enn først antatt. Derfor ble vinklingen på prosjektet endret slik at fokus ble på å lage et "Proof of Concept".

Vi vil først og fremst takke Stephen D. Wolthusen og Nils Kalstad Svendsen for en utfordrende og lærerik oppgave, samt for entusiasme og engasjement i løpet av prosjektperioden. Vi har fått god tilbakemelding og konstruktiv kritikk på både veiledningsmøtene vi har hatt på HiG og e-post vi har sendt i forbindelse med uforutsette problemer eller generelle spørsmål.

Vi vil også rette en generell takk til alle brukerne på jME-forumet for velvillig hjelp og veiledning på våre spørsmål. I tillegg vil vi også benytte anledningen til å takke alle som har korrekturlest rapporten for oss.

Gjøvik, 25. mai 2009

Robin Skilbrei Sveen

Nils Peter Lunden

Hans Jacob Koren Lund

Innhold

Sammendrag . . . . . **i**
Forord . . . . . **ii**
Innhold . . . . . **iii**
Figurer . . . . . **vi**
1 Introduksjon **1**
1.1 Organisering av rapporten 1
1.2 Kort om kritiske infrastrukturer 1
1.2.1 Gjensidig avhengighet 2
1.3 Problemstilling 2
1.4 Oppdragsbeskrivelse 3
1.5 Formål med prosjektet 3
1.6 Gruppens arbeidsformer 3
1.7 Gruppens bakgrunn og kompetanse 3
1.7.1 Rapportens målgruppe og hensikt 3
1.7.2 Rammer 4
1.8 Prosjektorganisering 4
1.8.1 Roller i gruppen 4
1.8.2 Grupperegler 5
1.8.3 Rutiner i gruppen 5
1.9 Planlegging, oppfølging og rapportering 5
1.9.1 Systemutviklingsmodell: Scrum 5
1.9.2 Daglige statusmøter 6
1.9.3 Sprint-planleggingsmøte 6
1.10 Utviklingsmiljø og kvalitetssikring 6
1.10.1 Utviklingsmiljø 6
1.10.2 Dokumentasjon og konfigurasjonsstyring 7
1.10.3 Risikoanalyse 8
1.11 Terminologi 8
2 Kravspesifikasjon **9**
2.1 Generelle krav 9
2.1.1 Omgivelser 9
2.1.2 Systemets brukere 9
2.1.3 Maskinkrav 9
2.1.4 Effekter og kontroll 9
2.1.5 Forskjellige modus 10
2.1.6 GUI 10
2.2 Flytdiagram for menygrensesnitt 11
2.3 Use Case 12
2.3.1 Use Case-diagram 12
2.3.2 Use Case-beskrivelse 13

2.4 Funksjonell kravspesifikasjon . . . 16
2.4.1 Funksjon . . . 16
2.4.2 Struktur . . . 16
2.5 Visualisering . . . 17
2.5.1 Oversvømmelser og lekkasjer . . . 17
2.5.2 Brann og brannspredning . . . 17
2.6 Demonstrasjonsscenarier . . . 17
2.6.1 Krav til sykehusscenario . . . 17
2.7 Utgivelser . . . 18
2.8 Rutiner for organisering og kvalitetssikring . . . 18
2.8.1 Dokumentasjon . . . 18
2.8.2 Backup . . . 18
2.8.3 Konfigurasjonsstyring . . . 18
2.8.4 Testing . . . 18
2.9 Supplementær kravspesifikasjon . . . 19
2.9.1 Videreutvikling . . . 19
2.9.2 Brukervennlighet . . . 19
3 Design . . . 21
3.1 Innledning . . . 21
3.1.1 Generelle mål . . . 21
3.1.2 Kort beskrivelse av systemet . . . 21
3.1.3 Bruksområde . . . 21
3.1.4 Rammer . . . 21
3.2 Overordnet design og systemarkitektur . . . 21
3.2.1 Beskrivelse av systemet og arkitekturen . . . 21
3.2.2 Klassediagram . . . 22
3.3 Design av brukergrensesnitt . . . 24
3.4 Design av modeller, effekter og scenarie . . . 24
3.4.1 Modeller . . . 24
3.4.2 Effekter . . . 26
4 Implementering, koding og produksjon . . . 27
4.1 Strukturen i hovedklassen . . . 27
4.2 Materialsystem . . . 28
4.3 Temperatur . . . 28
4.3.1 Vektorfelt . . . 29
4.3.2 Isoleringseffekt . . . 29
4.4 Brann . . . 30
4.5 Soteffekt . . . 31
4.6 Partikkelsystem . . . 32
4.6.1 Oppbygging av partikkelsamlinger . . . 34
4.6.2 Overflatepåvirkning av partikler . . . 35
4.6.3 Partikkelpåvirkning av vektorfelt . . . 36
4.6.4 Vektorfeltpåvirkning av partikler . . . 37
4.7 Gass-systemet . . . 38
4.7.1 Gassfelt . . . 38
4.7.2 Gasskontroller . . . 38

4.8 Oversvømmelse . . . 38
4.9 Slukking av brann . . . 40
4.10 Meldingsdisplay . . . 40
4.11 Musepekerverktøy . . . 41
4.12 Produksjon og modellering av scenarier . . . 42
4.12.1 Utførelse . . . 42
4.12.2 Scenarier for demonstrasjon . . . 43
4.12.3 Sykehus-scenario . . . 43
4.12.4 Kabelbrann-scenario . . . 43
5 Testing og kvalitetssikring . . . 45
5.1 Kvalitetssikring . . . 45
5.2 Testing . . . 45
6 Installasjon og realisering . . . 47
6.1 Innlasting av modeller . . . 47
6.2 Installasjon . . . 47
6.3 Eksekvering . . . 48
7 Beskrivelse av utviklingsprosessen . . . 49
7.1 Vår anvendelse av Scrum . . . 49
7.2 Utvikling av systeminkrementer . . . 49
7.3 Arbeidsfordeling . . . 49
8 Diskusjon av resultater . . . 51
8.1 Resultater . . . 51
8.2 Muligheter og valg underveis . . . 51
8.2.1 Valg av utviklingsspråk . . . 51
8.2.2 Valg av spillmotor . . . 52
8.2.3 Valg av utviklingsverktøy . . . 52
8.2.4 Valg av tekstbehandlingsverktøy . . . 52
8.3 Forslag til videre utvikling . . . 52
9 Evaluering av gruppens arbeid . . . 55
9.1 Innledning . . . 55
9.2 Organisering . . . 55
9.3 Fordeling av arbeidet . . . 55
9.4 Prosjekt som arbeidsform . . . 55
9.5 Tilbakeblikk på risikoanalysen . . . 55
9.6 Subjektiv opplevelse av bachelorprosjektet . . . 56
10 Konklusjon . . . 59
Bibliografi . . . 60

Figurer

1 Skjerminnstillinger . . . 10
2 Flytdiagram . . . 11
3 Use Case-diagram . . . 12
4 Nettverksarkitektur . . . 22
5 Konseptuelt klassediagram . . . 23
6 Navigasjonskart- meny . . . 25
7 Forslag til layout i hovedmeny . . . 25
8 Temperaturfeltet visualisert . . . 29
9 Vektorpåvirkning . . . 29
10 Temperaturøkning og brann . . . 31
11 Sot . . . 32
12 Sideforskyvning av sot . . . 32
13 Ytelsestest av partikkelsystem med jME Physics 2.0 . . . 33
14 Brann og røyk med overflatepåvirkning . . . 34
15 Prinsippskisse, overflatepåvirkning. . . . 36
16 Vektorfelt utgangspunkt . . . 37
17 Vektorfelt påvirket . . . 37
18 Tidlig eksplosjonsspredning . . . 39
19 Vannoverflate med refleksjon . . . 39
20 Feilmargin brannslukking . . . 40
21 Meldingsdisplay . . . 41
22 Tiltenkt funksjon for tekstkommunikasjon . . . 41
23 Skjermdump av blenders modelleringsmiljø . . . 42
24 Skjermdump av MonkeyWorld3D . . . 42
25 Sykehusscenario med serverrom, senger, ventilsjakt osv. . . . 43
26 Import-/eksportoversikt . . . 47

1 Introduksjon

1.1 Organisering av rapporten

Kapittel 1 - Innledning Beskrivelse av bakgrunn for prosjektet, kort om kritiske infrastrukturer, målgruppe, formål og arbeidsform, oppgavebeskrivelse, mål, rammer, organisering av gruppen, planlegging, konfigurasjonsstyring, utviklingsmiljø.

Kapittel 2 - Kravspesifikasjon Definering av hva som skal løses, krav og oppbygging.

Kapittel 3 - Design Design av implementasjonen

Kapittel 4 - Implementering, koding og produksjon Forklarer hvordan de forskjellige delene av implementeringen er løst. Metoder, valg og forutsetninger.

Kapittel 5 - Testing og kvalitetssikring Beskriver hvordan kode og funksjonalitet har blitt testet i løpet av prosjektperioden.

Kapittel 6 - Installasjon og realisering Forklaring av innlasting av modeller, installasjon og eksekvering.

Kapittel 7 - Beskrivelse av utviklingsprosessen Beskrivelse av hvordan vi har jobbet, vår bruk av systemutviklingsmodellen Scrum.

Kapittel 8 - Diskusjon av resultater Diskusjon av resultatet av implementasjonen og rapporten opp mot kravspesifikasjonen.

Kapittel 9 - Evaluering av gruppens arbeid Evaluering av gruppens arbeidsform, organisering og gjennomføring av oppgaven. Vurdering av tidsfrister og grad av oppfyllelse av oppgaven.

Kapittel 10 - Konklusjon Hovedkonklusjon av oppgaven.

Vedlegg Terminologi, fremdriftsplan, møtereferater, logg, statusrapporter og relevant korrespondanse.

1.2 Kort om kritiske infrastrukturer

Kritiske infrastrukturer blir ofte definert forskjellig fordi det i dag ikke finnes noen entydig definisjon på begrepet. Det brukes forskjellige definisjoner av land og organisasjoner over hele verden. Vi velger å legge NATOs (og trolig den mest tradisjonelle) definisjon til grunn. Denne er formulert slik:

> Critical infrastructure are those facilities, services and information systems which are so vital to nations that their incapacity or destruction would have a debilitating impact on national security, national economy, public health and safety and the effective functioning of the government.[19]

Dette innebærer de teknologiske nettverkene som understøtter samfunnet, herunder transportnett, energinett, elektroniske kommunikasjonsnett, vannledninger og avløp/kloakk. Felles for alle kritiske infrastrukturer er at de er knyttet til en eller flere knutepunkter. Dette kan være driftssentraler, transformatorstasjoner, terminaler osv. Disse knutepunktene vever ofte sammen de forskjellige samfunnsfunksjonene vi til daglig er avhengige av. Noen av de større kjente knutepunktene er i dag spesielt beskyttet av det norske lovverket ut i fra hensynet til landets sikkerhet og andre viktige samfunnsinteresser. Lovverket beskytter likevel kun spesielle knutepunkter for statlig virksomhet. Dette kan sies å være en svakhet, da stadig flere statlige oppgaver av samfunnskritisk art utføres via privateid infrastruktur. Det ville være en tilnærmet umulig oppgave å sikre alle knutepunkter og alle forbindelser i kritiske infrastrukturer fullstendig. I dagens samfunn er nærmest alle infrastrukturer knyttet sammen på en eller annen måte.

1.2.1 Gjensidig avhengighet

Vi har tre typer gjensidig avhengighet mellom kritiske infrastrukturer, som er relevante i denne sammenhengen: [18]

- **Fysisk avhengighet.** To infrastrukturer som hver for seg er direkte avhengig av det den andre produserer. *Eksempel: Et kullkraftverk er avhengig av at et tog kommer med daglige leveringer av kull. Toget, som drives av energi fra strømnettet, er avhengig av kraftverket for å få levert strøm til å drive det fremover. Slik er de begge fysisk avhengig av hverandre.*
- **Geografisk avhengighet** finner sted når deler av to eller flere kritiske infrastrukturer er fysisk nær hverandre. Dersom svikt i en av disse infrastrukturene skulle igangsette en brann eller en eksplosjon, vil dette mest sannsynlig føre til forstyrrelser eller brudd i de andre infrastrukturene. *Eksempel: To samfunnskritiske kabler som begge er deler av ulike infrastrukturer, er ført i samme kabeltrasé. Dette kunne være en strømførende kabel og en større fiberkabel. Skulle det oppstå en brann som følge av en kortslutning i den strømførende kabelen, er det nærliggende å tro at fiberkabelen ville blitt skadet som en følge av brannen. Dette kunne igjen føre til at deler av befolkningen ville mistet tilknytningen til kommunikasjonsnett. Dette kan være alt i fra private husstander, til sykehus, politistasjoner, større bedrifter osv.*
- **Cyber-avhengighet.** Dersom tilstanden til en kritisk infrastruktur er avhengig av informasjon overført via internett. *Eksempel: SCADA-systemer (Supervisory Control And Data Acquisition) er brukt for å kontrollere og overvåke industrielle prosesser som vannrensing, raffinering og energiproduksjon. SCADA-systemene er på sin side avhengig av elektrisk kraft, produsert fra kraftverk de selv overvåker og kontrollerer.*

I denne prosjektoppgaven vil vi først og fremst konsentrere oss om geografisk avhengighet mellom kritiske infrastrukturer i mindre skala.

1.3 Problemstilling

Kritiske infrastrukturer er systemer som er av høy viktighet for samfunnet. Forstyrrelser av disse infrastrukturene kan føre med seg alvorlige samfunnsmessige og/eller økonomiske konsekvenser. Dette er et tema som er av spesiell interesse for informasjonssikkerhetsavdelingen ved Høgskolen i Gjøvik. Vi har i denne sammenheng fått i oppdrag å utvikle et visualiseringsverktøy som gjør det mulig for brukere å generere forskjellige scenarier

basert på brukerens spesifikasjoner.

1.4 Oppdragsbeskrivelse

I prosjektbeskrivelsen etterlyses et verktøy som gjør det mulig å visualisere ulike scenarier. Eksempelvis branner, eksplosjoner og oversvømmelser i tilknytning til, eller som følge av forstyrrelser i, kritiske infrastrukturer. Det skal også være mulig å inspisere potensielle faremomenter i en gitt bygningskonstruksjon fra forskjellige perspektiv/synsvinkler, dvs. at brukeren skal kunne bevege seg fritt innenfor bygningens vegger jfr. FPS-spill. Utover dette skal man kunne importere konstruksjonsdata fra CAD-format eller et annet standardformat, samt brukerskapte scenarier på en eller annen måte. Vi anser dette for å være helt avgjørende for hvorvidt verktøyet blir tatt i bruk eller ikke. Dersom ressursene strekker til er det også ønskelig at man skal kunne gå igjennom forskjellige scenarier synkront over nettet med en annen part. For å kunne utvikle et slikt verktøy blir det nødvendig å benytte et eksisterende rammeverk i form av en spillmotor.

1.5 Formål med prosjektet

Vi ønsker i første omgang å vise at det er mulig å sette sammen et verktøy for visualisering av svikt i kritiske infrastrukturer som følge av gjensidig avhengighet, vha. fri programvare. Vi har som målsetting at dette verktøyet skal ha en bruksverdi som et analyseverktøy der man kan laste inn bygningsmodeller, og sette opp scenarier i disse. Slik skal man kunne vurdere faremomenter, samt demonstrere viktige prinsipper i forhold til gjensidig avhengighet mellom kritiske infrastrukturer.

1.6 Gruppens arbeidsformer

Utviklingen av systemet har i all hovedsak foregått ved at vi har fordelt ansvaret for modulene i inkrementene mellom gruppemedlemmene. Ved større og mer komplekse moduler har vi samarbeidet for å kunne løse dette raskere. Grov bestemmelse av hvordan ting skal fungere har blitt diskutert i fellesskap slik at alle har en grunnleggende forståelse av hver del av systemet.

1.7 Gruppens bakgrunn og kompetanse

Samtlige gruppemedlemmer har fulgt studiet bachelor i ingeniørfag - data, med spesialisering innen programvareutvikling. Vi har fra før god kjennskap til programmering innen C++/C og Java. Likevel hadde ingen av oss noen erfaring med spillprogrammering eller 3D-modellering i forkant av dette prosjektet.

1.7.1 Rapportens målgruppe og hensikt

Målgruppen for denne rapporten består hovedsakelig av veilederen for prosjektet, oppdragsgiver og ekstern sensor. Oppdragsgiveren i vårt tilfelle har vært Høgskolen i Gjøvik v/Nils Kalstad Svendsen og Stephen D. Wolthusen. Andre mottakere er studenter ved Høgskolen i Gjøvik som skal ta for seg et liknende hovedprosjekt eller som ønsker å studere arbeidet i andre sammenhenger.

Denne rapporten har til hensikt å gi en fordypende beskrivelse av prosjektet for samtlige mottakere. Prosjektet skal etterhvert vurderes av veileder og ekstern sensor, og det er derfor viktig for oss å få frem hva som er blitt gjort, og hvilke vurderinger og avgjørelser som er tatt underveis. Dette er også viktig med tanke på videreutvikling av

programvaren. Utover det nevnte, fungerer rapporten også som et hjelpeverktøy for oss i gruppen under prosjektarbeidet.

1.7.2 Rammer

For å holde prosjektet åpent og juridisk uproblematisk å videreutvikle, har vi benyttet oss av et utviklingsmiljø og en spillmotor under lisens for åpen kildekode. Det har ikke blitt satt krav til ytelse og detaljrikhet i scenariene som genereres av verktøyet. På den andre siden trenger vi heller ikke forholde oss til begrensede maskinvareressurser. Utover lisensproblematikken er det ikke annet enn den gjeldende tidsrammen for bacheloroppgaver ved Høgskolen i Gjøvik som setter reelle begrensninger.

1.8 Prosjektorganisering

1.8.1 Roller i gruppen

- Nils Peter Lunden - Prosjektleder, kommunikasjonsansvarlig, backupansvarlig.
- Hans Jacob Koren Lund - Webmaster, dokumentasjonsansvarlig.
- Robin Skilbrei Sveen - Testansvarlig, buildmaster.

Disse rollebeskrivelsene har blitt hentet ut i fra forprosjektrapporten, for å gi et innblikk i hvordan rollene ble definert i forkant av utviklingen.

Prosjektleder Har ansvar for å ta tak og være motivator dersom større problemer oppstår og noen blir stående fast lenge med et problem. Derimot er det ikke lederens jobb å kjøre sololøp og ta avgjørelser uten medhold i gruppa. Skulle sterk uenighet oppstå når en avgjørelse skal tas, vil det være lederen som må ta avgjørelsen.

Kommunikasjonsansvarlig Har ansvar for å avtale møter med oppdragsgiver og veileder og er generelt gruppens kontaktperson.

Backupansvarlig Har ansvar for å ta backup av subversion-området i slutten av hver sprint. Dette skal brennes ut på CD/DVD og merkes med dato.

Webmaster Har ansvar for å sette opp og vedlikeholde websidene tilknyttet prosjektet. Han vil også ha ansvar for å publisere en oppdatering på prosjektets progresjon hver gang ny funksjonalitet er på plass i programvaren som skal lages. Andre medlemmer av gruppen vil også ha muligheten til å publisere ting på sidene, men webmaster har hovedansvaret for at oppdateringer skjer.

Dokumentasjonsansvarlig Har det overordnede ansvaret for dokumentasjonen og å sette sammen de ulike delene som blir forfattet av gruppemedlemmene til en helhet. Han har også ansvaret for å skrive et kort referat av møtene med oppdragsgiver og/eller veileder.

Testansvarlig Har hovedansvaret for å utføre tester, eventuelt sørge for at det som skal testes blir testet og at det fungerer slik som det skal. Han har også ansvaret for at tester blir dokumentert.

Buildmaster Denne personen skal til enhver tid ha siste versjon av all kildekode slik at vi ser at koden kompilerer og at de ulike delene virker sammen.

Da vi har operert med empirisk testing, har testansvaret mer eller mindre blitt delt mellom alle gruppemedlemmene. Ellers har ansvarsfordelingen vært som vi planla i utgangspunktet.

1.8.2 Grupperegler

I begynnelsen av prosjektet ble det utarbeidet et sett grupperegler, dette er reglene i sin opprinnelige form:

- Uenigheter løses ved avstemning.
- Prosjektleder blir valgt ved prosjektets start og fungerer ut hele perioden.
- Gyldig fravær skal meldes så tidlig som mulig. Gjentakende umeldt fravær vil kunne resultere i utestengelse og avkreditering.
- Alle tar ansvar for at alle mål blir nådd.
- Alle deltar på gruppe- og statusmøter så langt det lar seg gjøre. Hvis man ikke kan møte opp skal dette gis beskjed om.

1.8.3 Rutiner i gruppen

- Beskrivelse av arbeidet og arbeidstimer er blitt ført i arbeidsloggen av hver enkelt, hver dag.
- Det er blitt skrevet møtereferater, der alle beslutninger og avtaler er notert i detalj.
- Vesentlige beslutninger er blitt dokumentert.
- Det har blitt tatt backup av hele repositoriet (rapport, kildekode, og øvrig dokumentasjon) hver fredag.
- Hver enkeltfil (tekst eller kode) har hatt én ansvarlig person. Dette har medført at vi har unngått konflikter og unødvendig hodebry i forhold til opplasting til repository.

1.9 Planlegging, oppfølging og rapportering

1.9.1 Systemutviklingsmodell: Scrum

Ingen i prosjektgruppen har jobbet med et spillmotor-rammeverk tidligere, og har slikt sett vanskelig å estimere hvor mye vi klarer å produsere, tatt i betraktning den tiden vi har hatt til rådighet. Det har derfor vært viktig for oss å sørge for at vi alltid har hatt et synlig sluttprodukt som kan demonstreres. Etter anbefaling fra veileder, konkluderte vi med at vi ville satse på en agile utviklingsmodell. Tidligere i systemutviklingsfaget ble det av agile-metoder stort sett fokusert på eXtreme Programming(XP) og Scrum. Vi valgte derfor å eliminere andre grener av agile-metoder. Siden vi er en gruppe på tre, er parprogrammering (XP) umulig, og siden oppdragsgiver ikke er tilgjengelig til envher tid lar heller ikke “on-site customer-prinsippet” (XP) seg gjøre i praksis. Med dette som grunnlag falt valget på scrum-modellen. Scrum legger til rette for en iterativ og inkrementell arbeidsmåte, og sørger for at vi ikke står uten et presentabelt produkt. Scrum-modellen legger opp til arbeidsintervaller (sprinter) på eksempelvis to til fire uker. Ved hver sprintslutt skulle vi ha produsert et kjørbart produkt. Dette innebærer at vi til enhver tid har hatt et produkt som er demonstrerbart/presentabelt. Dette har også bidratt

til at vi som utviklere i gruppen har hatt noe visuelt og håndfast å forholde oss til. Vi i gruppen føler at prosjektet er nokså ambisiøst med tanke på arbeidsmengde, og det har derfor vært motiverende at vi med scrum-modellen har kunnet dele opp prosjektet i mindre og mer kortsiktige delmål. Scrum-modellen er ikke spesielt tidkrevende å sette seg inn i, og vi har dermed sluppet å bruke alt for mye tid på å sette opp rammeverket for arbeidsmetodikken. Vi har ikke ansett det som nødvendig å innføre noen nye rollebegreper i sammenheng med innføringen av scrum i en såpass liten utviklergruppe. Scrum-modellen har ikke nødvendigvis blitt fulgt slavisk på alle punkter, men den har fungert som grunnsteinen i strukturen, planleggingen og arbeidsmetodikken i prosjektet.

1.9.2 Daglige statusmøter

Deltakere: utviklergruppen.
I henhold til scrum-modellens retningslinjer er det blitt holdt et femten-minutters statusmøte til et fast tidspunkt hver arbeidsdag. Her fortalte alle gruppemedlemmer hva de har gjort siden siste statusmøte og hva de planla å gjøre frem til det neste. Andre problemstillinger ble også tatt opp på dette møtet. Vi har valgt å være fleksible på møtenes varighet, da det av og til har vært viktige/vanskelige problemstillinger å diskutere.

1.9.3 Sprint-planleggingsmøte

Deltakere: utviklergruppen, veileder, oppdragsgiver.
Under dette møtet som er blitt holdt etter hver sprint, så langt det har latt seg gjøre, har vi sett tilbake på hva som ble gjort, hva som gikk bra og evt. hva som gikk galt. Dersom oppdragsgiver har ønsket å gjøre endringer i prioriteringer eller justeringer i kravspesifikasjonen (scrum-modellen er åpen for endringer i kravspesifikasjonen underveis) har det blitt tatt opp på disse møtene. Ut over dette, har møtedeltakerene diskutert neste forestående sprint og tatt opp eventuelle problemstillinger og innspill. Dokumentasjonsansvarlig i gruppen har notert det som er blitt diskutert under møtet, og ført inn alle møtereferater.

1.10 Utviklingsmiljø og kvalitetssikring

1.10.1 Utviklingsmiljø

Følgende programvare og biblioteker kommer vi til å bruke i arbeidet vårt:

jMonkeyEngine [8] Spillmotoren vi har brukt, versjon 2.0. Dette er et såkalt Java scenegraph API. Forkortes til jME. Utgitt under BSD-lisensen [5].

JGN [9] Java game networking. Java-rammeverk for nettverksspill.

jME Physics [10] Bibliotek for fysikk som er tett knyttet til jME.

JavaDoc Standard verktøy for dokumentering av java-kildekode.

Eclipse [11] Java IDE som er brukt til å skrive all kode. Her har vi også brukt en programutvidelse for å integrere Subversion-funksjonalitet. Denne har fungert som et bindeledd mellom Eclipse og jME, JGN og jME Physics 2.

Eclox hot Doxygen-utvidelse til Eclipse.

Blender [7] 3D-modellereringsverktøy som vil bli brukt til å konvertere modellfiler.

LEd [6] Tekstredigeringsprogram for LaTeX. LEd produserer også selvfølgelig DVI, Postscript og PDF med enkle grep.

MS Project 2007 Vil bli brukt til å legge tidsplan og å lage Gantt-skjema.

Monkey World 3D [15] Dette programmet brukes for å endre på modeller og modellers nodestruktur. Det lagrer modeller i et format som er klart for å brukes i jMonkeyEngine.

I forbindelse med forprosjektet, ble det planlagt å bruke disse verktøyene:

Apache Ant [12] Project build tool som kan integreres i Eclipse.

Doxygen [13] Program brukt til å generere bla. LaTeX-dokumentasjon basert kildekode.

Disse har ikke blitt benyttet i prosjektet, siden vi under hele implementeringsfasen har hatt god oversikt over kildekoden og dokumenteringen av denne (JavaDoc). Apache Ant ville vært mer aktuell å introdusere ved større kodemengder, slik vi ser det.

1.10.2 Dokumentasjon og konfigurasjonsstyring

All dokumentasjon som er produsert i forhold til rapporten er blitt skrevet i LaTeX for å forenkle samarbeidet om dokumenter. Dokumentene består av ren tekst, noe som har muliggjort versjonshåndtering med Subversion. Dette har sørget for at at vi har kunnet fokusere mye mer på dokumentasjonens innhold fremfor på utseendet. Kommentering er blitt skrevet mens vi har kodet, og aldri i etterkant. Når det gjelder skriving av Java-kildekode generelt har vi støttet oss til de retningslinjer og konvensjoner som Sun baserer seg på [4]. Videre har vi konsekvent brukt tab-tasten som innrykk. 2-4 whitespaces per tab gir et godt utseende på koden.

1.10.3 Risikoanalyse

Risiko	Alvorlighet	Sannsynlighet	Tiltak
Langvarig sykdom.	Middels	Lav	Skrive oversiktlig og veldokumentert kildekode slik at det ikke blir problematisk for en annen på gruppa å ta over ansvarsområdet til den som måtte bli syk.
Svikt i maskinvare.	Høy	Lav	Bruke subversion-området til lagring av alt prosjektrelatert. I tillegg skal det tas sikkerhetskopi av dette innholdet i slutten av hver sprint. Hver og en sørger for at sin PC er sikret, det vil si at operativsystem er oppdatert og sikret med antivirusprogramvare.
Ikke bli ferdig med funksjonaliteten som skal leveres på sprinten.	Lav	Middels	Ta igjen tapt tid på sprinter som omfatter mindre kompleks funksjonalitet. I verste fall fire på kravene til funksjonalitet da dette er et forskningsprosjekt.
Manglende kompetanse.	Lav	Middels	Lese teori og tilegne oss kompetanse underveis. Bruke veileder og oppdragsgiver aktivt i prosessen.
Buggy kode.	Lav	Høy	Utnevne en på gruppa til testansvarlig. Foreta grundig testing når ny funksjonalitet blir lagt til. Prioritere bug-fiksing fremfor å kode ny funksjonalitet.
Overskride prosjektets tidsfrist.	Høy	Lav	Å bruke vår tidsplan (Gantt-skjema) aktivt og foreta justeringer i forhold til den underveis. Fokusere på å delegere oppgaver slik at ikke mer enn én person jobber med en oppgave.
Ende opp uten et demonstrerbart produkt.	Høy	Lav	Velge utviklingsmetode som krever periodevis innlevering av kjørbar programvare.

Generelt tiltak: Utarbeide et godt forprosjekt og bruke dette aktivt videre i arbeidsprosessen for å øke kvaliteten på arbeidsmetoder og resultat.

1.11 Terminologi

Vi har forsøkt å finne gode og dekkende norske ord i de tekniske forklaringene i rapporten. Det vil allikevel forekomme en god del ord som i større eller mindre grad ikke lar seg oversette. Vi har derfor lagt ved en liste over terminologien og en enkel forklaring til hvert av disse uttrykkene i vedlegg A.

2 Kravspesifikasjon

2.1 Generelle krav

Kravspesifikasjonen ble skrevet før vi begynte med selve utviklingen av applikasjonen. I dette utviklingsprosjektet kan kravspesifikasjonen sees på som et "levende" dokument som kommer til å bli revidert i løpet av utviklingsperioden.

2.1.1 Omgivelser

Vi vil i begynnelsen konsentrere oss om å sette opp en enkel modell slik at vi kommer fort i gang med å lage kjernefunksjonalitet: brann-, vann- og gasseffektene. Etter hvert modellerer vi en større og mer reell bygningsmodell. Modellen bør inneholde de fleste typer materialer og infrastrukturer, slik at vi får testet all funksjonalitet hver gang det testes. Denne skal være et produkt av Blender og Monkey World 3D som er veien å gå for å få bygningsmodeller inn i spillmotoren.

2.1.2 Systemets brukere

Dette systemet utvikles først og fremst med oppdragsgiver som antatt bruker, men det vil etter hvert være aktuelt at sikkerhetseksperter, bedriftsledelse og eventuelt andre interesserte parter tar i bruk applikasjonen.

2.1.3 Maskinkrav

Applikasjonen skal kunne kjøre tilfredsstillende på en, etter dagens standard, moderne stasjonær datamaskin. Da programvarens målgruppe vil være mellomstore og store bedrifter, stilles det utover dette ingen spesielle krav. Dette fordi vi anser at de er aktuelle brukere av programvaren har tilgang på utstyr som er kraftig nok til å kjøre det. Vår referansemaskin har følgende spesifikasjoner:

- Intel Quad Core 2.2 GHz
- 2GB RAM
- ATI Radeon 4850
- Windows Vista 64

2.1.4 Effekter og kontroll

Det er ingen krav til at de synlige effektene i programmet skal være veldig gode, det er viktigere at disse er en god tilnærming til virkeligheten enn at de ser pene ut. Det er ikke nedsatt noen krav til lyd og lydeffekter, dette er ene og alene en visualisering og det skal ikke være med noen form for lydeffekter. Dynamiske lyseffekter skal ikke være en del av løsningen, alt av omgivelser og objekter skal være fullstendig synlig til enhver tid. Når brukeren navigerer rundt i en modell, gjøres det ved hjelp av mus og tastatur. Når det er flere brukere inne i et scenario skal de kunne se hverandres posisjon - hver bruker skal ha en visuell representasjon/avatar. Dette bør være noe enkelt, for eksempel en liten kule eller boks med brukerens navn over. Kollisjonsdeteksjon skal ikke brukes i løsningen, det er et poeng at brukerne skal kunne inspisere alle deler av et scenario. Brukeren skal altså

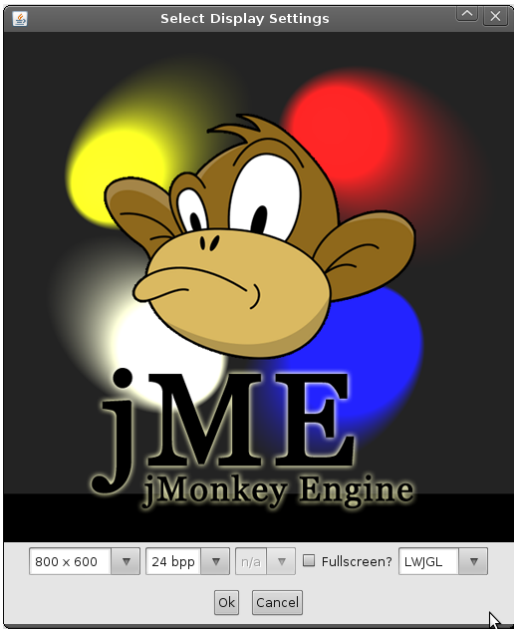
"fly" rundt i omgivelsene og ikke bli trukket ned på underlag som om det skulle være tyngdekraft tilstede.

2.1.5 Forskjellige modus

Brukeren skal ha mulighet til å kunne sette opp en server, koble seg til en annen server eller sette opp et scenario lokalt på sin egen maskin. Det skal kunne kobles til en server via LAN eller Internett. I oppsett av server, både felles og lokal, kan brukeren først gå rundt i bygningsmodellen å velge hvor hendelser skal starte. Alle brukerne vil ha muligheten til å gå rundt i bygningsmodellen hvor de kan observere hva som skjer de stedene det er igangsatt kritiske hendelser.

2.1.6 GUI

Rett etter programstart skal det vises et grafisk brukergrensesnitt der brukeren kan velge skjermoppløsning, fargedybde, fullskjerm/ikke fullskjerm (figur 1). Dette er standard for alle spill og programmer skrevet med jMonkeyEngine. Applikasjonen skal så ha en hovedmeny som presenterer hvilke valg brukeren har. Dette skal være en meny som blir presentert etter foregående meny, men *før* programmet starter å laste inn modell og man kan bevege seg rundt i modellen.



Figur 1: Her stilles skjerminnstillingene.

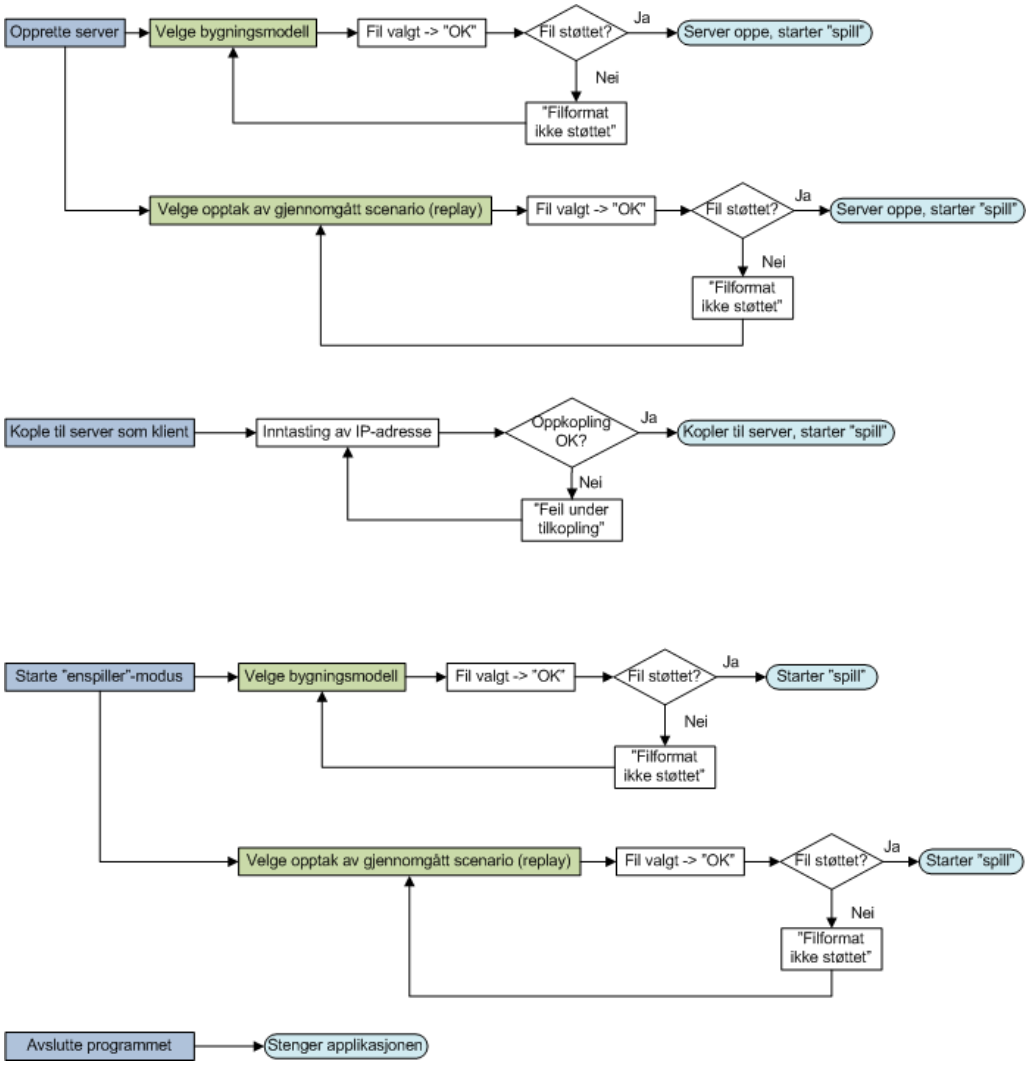
Disse valgene skal være tilgjengelige i hovedmenyen:

- Opprette server
  - Velge bygningsmodell
  - Velge opptak av gjennomgått scenario, "replay"
  - Start "spill"
- Koble til server som klient

  - Skrive inn ip-adresse til server
  - Start “spill”
- Starte “enspiller”-modus
  - Velge bygningsmodell
  - Velge opptak av gjennomgått scenario, “replay”
  - Start “spill”
- Endre “spillernavn”
- Avslutte programmet

2.2 Flytdiagram for menygrensesnitt

Figur 2 viser hvordan flyten i hovedmenyen beskrevet i kapittel 2.1.6 skal foregå basert på brukerens valg. Den beskriver også hvordan feilsituasjoner skal påvirke gangen i menyen.



Figur 2: Flytdiagram.

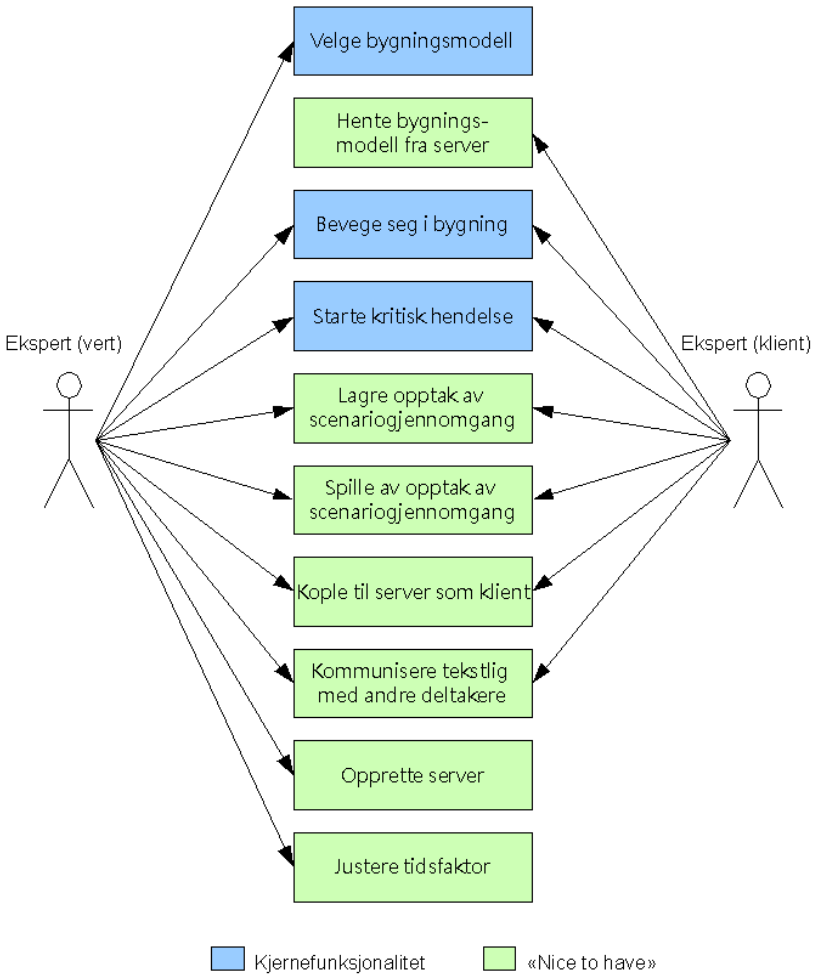
2.3 Use Case

Use case er en detaljert beskrivelse av ulike hendelser i systemet. Det blir beskrevet hvordan det er tenkt at handlingsgangen skal være, og eventuelt hvordan eventuelle feil som oppstår skal behandles. De inneholder også informasjon om hvem som skal ha tilgang til å utføre hendelsen, og hvilken type hendelse det er dreier seg om.

Årsaken til at vi lager disse beskrivelsene er at det blir lettere for oss å forstå hvordan de forskjellige delene skal fungere når de skal implementeres. I tillegg vil dette fungere som en referanse i testingen. Vi kan da sjekke om funksjonaliteten som er implementert samsvarer med den ønskede funksjonaliteten.

2.3.1 Use Case-diagram

Denne figuren viser en oversikt over forskjellige handlinger og valg brukeren skal ha i systemet. Hvert valg blir presentert som en individuell "case" som beskriver brukerne det gjelder og har en inngående forklaring på handlingsforløpet. Figur 3 viser hvilke handlinger/valg som er viktigst i dette systemet (de mørke rektanglene) og det er disse vi har blitt enig med veileder og oppdragsgiver om å fokusere på.



Figur 3: Use Case-diagram

2.3.2 Use Case-beskrivelse

Velge Bygningsmodell:
Aktør: Ekspert (vert)
Type: Konfigurering
Beskrivelse: Brukeren får muligheten til å laste bygningsmodell, enten ved oppretting av ny server, eller ved kjøring av enspillermodus. Brukeren får muligheten til å navigere filsystemet, for så å velge ut en ønsket bygningsmodell.
Variasjoner: Formatet er ikke støttet, og brukeren vil få beskjed om dette. Brukeren blir deretter returnert tilbake til undermenyen.
Betingelser: Bygningsmodell må være i .jme format.

Starte kritisk hendelse:
Aktør: Ekspert (vert og klient(er))
Type: Konfigurering
Beskrivelse: Brukeren skal kunne gå rundt i bygningen og sette opp en eller flere kritisk(e) hendelse(r) i form av brann, oversvømmelse, gass eller røyk. Verten kan hente frem musepekeren ved en ønsket posisjon og markere hendelsens utspringspunkt. Det skal også være mulig for brukeren å velge type hendelse som skal igangsettes. Dette kan gjøres ved et forvalg på tastaturet.
Variasjoner: Brukeren er koblet til som klient, og kan dermed ikke sette opp en kritisk hendelse. Melding på skjermen om at brukeren ikke har tilgang til følgende valg.
Betingelser: Kun verten har mulighet til å legge inn kritiske hendelser.

Bevege seg i bygning:
Aktør: Ekspert (vert og klient(er))
Type: Handling i spill
Beskrivelse: Brukeren skal ha mulighet til å gå rundt inne i bygningsmodellen og fritt kunne observere alle områder og hendelser. Det kreves at det benyttes følgende oppsett for forflytning:

- **W** - fremover
- **S** - bakover
- **A** - venstre
- **D** - høyre
- **Q** - opp
- **Z** - ned

Musen skal brukes for å endre synsvinkel, noe som tilsvarer et typisk spill-oppsett. Dette er en fordel da mange brukere allerede har et intuitivt forhold til å bevege seg i tre dimensjoner på denne måten.
Variasjoner: Det er ikke lastet inn en bygningsmodell i systemet. Brukeren får beskjed om at det må lastes inn en bygningsmodell, og sendt tilbake til hovedmenyen.
Betingelser: Bygningsmodell må være lastet inn.

Lagre opptak av scenariogjennomgang:
Aktør: Ekspert (vert og klient(er))
Type: Handling i spill
Beskrivelse: Brukeren skal kunne ta opp sin bevegelse og sine observasjoner under en scenariogjennomgang. Dette er ment å være til nytte ved presentasjoner, risikoanalyser etc. Brukeren skal her kunne igangsette opptaksmodus ved å gjøre et tastetrykk. Deretter skal brukeren få tekstlig beskjed på skjermen om at opptaket er i gang. Dette opptaket fortsetter å gå til brukeren stopper det eller ved en maksgrense bestemt av kapasitet i minne/lagringsplass. Både forflytning og endring av synsvinkel skal lagres. Det skal ikke være synlig forskjell på opptaket fra de opprinnelige bevegelsene utført av brukeren under opptaket.
Variasjoner: Det er ikke tilstrekkelig med lagringsplass på maskinen til å kunne lagre opptaket. Brukeren får ikke startet opptaksfunksjonen, og får beskjed om at det ikke er ledig plass på disken.
Betingelser: Tilstrekkelig lagringsplass/minne.

Spille av opptak av scenariogjennomgang:
Aktør: Ekspert (vert og klient(er))
Type: Handling
Beskrivelse: Brukeren skal kunne spille av opptak av registrerte bevegelser fra en tidligere scenariogjennomgang. Brukeren får muligheten til å gjøre dette i menyen, før spillet kjører. Her kan han/hun navigere filsystemet og velge et opptaksfil. Når brukeren velger filen han/hun vil laste inn, starter avspilling automatisk dersom filformatet er støttet.
Variasjoner: Filformatet brukeren har valgt er ikke støttet av systemet. Brukeren får beskjed om at formatet ikke støttes, samt en liste over aktuelle filformat som er støttet.
Betingelser: Filformat må være støttet

Kople til server som klient:
Aktør: Ekspert (vert og klient(er))
Type: Handling
Beskrivelse: Klienten(e) skal kunne kople til en server for å gå igjennom et scenario med andre eksperter. Verten som har satt opp serveren kopler til egen server lokalt. Dersom en klient skal kople til en ekstern server, skal han/hun kunne gjøre dette i menyen i forkant av kjøring. Her får brukeren opp et tekstfelt for inntasting av serverens IP-adresse. Når brukeren så taster enter-tasten, skal oppkoplingen begynne, og brukeren får følgende tilbakemeldinger i forhold til oppkoplingens status:

- Under oppkopling: “Kopler til 192.168.1.101...”
- Vellykket oppkopling: “Tilkopling vellykket.”
- Feil under oppkopling: “Tilkopling feilet.”

Dersom tilkoplingen er vellykket får brukeren melding om dette i grensesnittet under kjøring. Dersom tilkoplingen feiler, returneres brukeren til undermeny.
Variasjoner: Brukeren har ikke tilgang LAN/WAN-tilgang på sin maskin. Dukker opp feilmelding om manglende LAN/WAN-tilgang.

Betingelser: LAN/WAN-tilgang hos alle parter.

Kommunisere tekstlig med andre deltakere:
Aktør: Ekspert (vert og klient(er))
Type: Handling
Beskrivelse: Brukeren skal kunne kommunisere tekstlig med alle andre deltakere. Meldingene overføres i sanntid, og kommer opp på skjermen hos alle deltakere. Ved å gjøre et fordefinert tastetrykk skal brukeren få muligheten til å taste inn en tekstmelding via et tekstfelt som blir synliggjort nederst i høyre hjørne. Når brukeren trykker ned enter-tasten på tastaturet skal meldingen distribueres til alle tilkoplede klienter.
Variasjoner: Brukeren har ikke tilgang LAN/WAN-tilgang på sin maskin. Dukker opp feilmelding om manglende LAN/WAN-tilgang.
Betingelser: LAN/WAN-tilgang hos alle parter

Opprette server:
Aktør: Ekspert (vert)
Type: Konfigurering
Beskrivelse: Brukeren skal ha mulighet for å sette opp en server som andre eksperter kan kople seg til. Serveren skal kunne settes opp på en vanlig arbeidsstasjon. Brukeren skal enkelt kunne sette i gang denne serverprosessen via menyen som kommer opp før kjøring. Når serveren er i gang får brukeren presentert sin egen globale IP-adresse via grensesnittet under kjøring slik at denne kan distribueres til andre brukere som ønsker å kople opp som klienter mot serveren.
Variasjoner: Brukeren har ikke tilgang LAN/WAN-tilgang på sin maskin. Dukker opp feilmelding om manglende LAN/WAN-tilgang, og returneres til hovedmenyen.
Betingelser: LAN/WAN-tilgang.

Hente bygningsmodell fra server:
Aktør: Ekspert (klient)
Type: Handling
Beskrivelse: En klient må ha samme modell som verten for å kunne delta i gjennomgangen av scenariet. Klienten må i denne sammenheng kunne laste ned aktuell bygningsmodell fra serveren (verten). Når klienten kopler opp mot serveren sendes det informasjon om modellen fra serveren. Det skal så kjøres en sjekk for om denne modellen allerede eksisterer på klientens system. Dersom dette ikke er tilfellet, overføres modellfilen til klientens system, og brukeren på klienten informeres om dette via et tekstfelt i grensesnittet. Etter overføring kobles klienten på som vanlig.
Variasjoner: Brukeren har ikke tilgang LAN/WAN-tilgang på sin maskin. Dukker opp feilmelding om manglende LAN/WAN-tilgang, og returneres til hovedmenyen.
Betingelser: LAN/WAN-tilgang.

Justere tidsfaktor:
Aktør: Ekspert (vert)
Type: Handling
Beskrivelse: Sette opp eller ned tidsfaktoren som avgjør hvor mye fortere/langsommere tiden går. Dette har innvirkning på f.eks. hvor raskt et opptak avspilles/rekonstrueres.

Hvis man bare inspiserer en modell uten avspillingsfunksjonen vil tidsfaktoren påvirke hvor fort/langsomt varmen spres og alle andre ting som påvirkes av en tidsfaktor.
Variasjoner: Brukeren er koblet til som klient, og kan dermed ikke endre tidsfaktoren. Melding på skjermen om at brukeren ikke har tilgang til følgende valg.
Betingelser: Må være i "enspiller"-modus eller være vert.

2.4 Funksjonell kravspesifikasjon

2.4.1 Funksjon

Det vil bli laget et verktøy for visualisering av kritiske infrastrukturer, og deres innvirkning på hverandre. Oppnådde resultater vil bli vist frem i form av en video eller en livedemonstrasjon. Systemet skal inneholde funksjonalitet for materialvalg, brann, vann og gass. Brukeren skal ha muligheten til å bevege seg rundt for å observere hva som skjer.

2.4.2 Struktur

Applikasjonen kan naturlig deles inn i følgende underdeler:

Materialsystem: Skal tilegne alle elementene i bygningsmodellen et bestemt type materiale. Materialtypen inneholder informasjon om hvordan materialet vil oppføre seg ved ytre påvirkninger som brann, vann og trykk.

Fysikksystem: Fysikksystemet tar for seg hvordan elementene i rommet blir påvirket av brann, vann og gass. Temperaturforandringer og faktorer som trekk/vind i rommet håndteres i dette systemet. Dette skal ikke forveksles med jME Physics, da dette ikke inneholder slik funksjonalitet.

Partikkelsystem: Definerer egenskaper som røyk, brann, gass og eksplosjon. Disse partiklene skal visualiseres på en slik måte at det ikke er noen tvil om hva de representerer. Det skal være mulig å enkelt definere farge, størrelse, tekstur, utspringspunkt, og emisjonsretning i koden. Dette er spesielt viktig med tanke på videreutvikling av verktøyet. Partiklene skal holdes inne i rommet som omgir utspringspunktet, og de skal bli påvirket av luftsirkulasjon i omgivelsene.

Nettverkssystem: Skal gjøre det mulig å sette opp en server med et scenarie og tillate andre å koble seg til denne. Enten via Internett eller via lokalt nettverk. På denne måten kan man sammen inspisere scenarier selv om deltakerne er spredt over et stort geografisk område. Det har i oppgave å koordinere posisjonene til de forskjellige deltakerne i modellen og brukergenererte hendelser: tekstkommunikasjon, intruffet kritisk hendelse, brukere forlater/blir med i "spillet."

Meldingssystem: Brukeren har mulighet til å skrive inn en melding til andre personer innenfor samme serveren. Alle meldingene som blir sendt og mottatt blir presentert i meldingsfeltet. Når en kritisk hendelse inntreffer kommer også dette opp som en skriftlig melding i samme konsoll. Dette skal også holde på og lagre hendelsesforløpet slik at dette kan gjenskapes senere - replay-funksjon.

Konverteringssystem: Konverteringssystemet skal gjøre det mulig å laste inn bygningsmodeller i spillmotoren fra for eksempel en CAD-fil. Blender og Monkey World

3D må sees på som trinn i denne konverteringsprosessen og utgjør i realiteten konverteringssystemet. Det blir ikke foretatt noen konverteringer etter Monkey World 3D, modellfilen er da helt klar til å leses inn i spillmotoren.

2.5 Visualisering

Den aller viktigste modulen i dette prosjektet er selve visualiseringen. Dette er selve fundamentet i verktøyet. Det skal være mulig å visualisere oversvømmelser og lekkasjer, samt brann, eksplosjoner og røykutvikling. Beregningene som ligger til grunn for visualiseringen skal alle skje i sanntid, ikke på forhånd slik som det gjøres i NISTs FDS [20]. Der tar utregningene for ett minutts visualisering av et brann-scenario åtte timer å utføre (på en ytelsesmessig svak maskin etter dagens standard) [21].

2.5.1 Oversvømmelser og lekkasjer

Oversvømmelser skal visualiseres vha. et plan (væskeoverflate) eller ved bruk av et partikkelsystem (fullstendig væskemasse). Her må det tas en vurdering basert på spillmotorens ytelsesmessige begrensninger, og det må tas estetiske hensyn. Nivåstigningen bestemmes ut fra hvor stort volum av væske som blir tilført et rom per sekund. Ved spesielt høye vannivåer er det aktuelt at svake punkter som vinduer og dører gir etter. Dette er likevel ikke en prioritet, da det kun er i veldig sjeldne tilfeller det er aktuelt å analysere slike hendelser. Vanntrykket som kreves for at et vindu skal gi etter vil først oppnås ved meget høye vann-nivå; vesentlig høyere enn rommets høyde.

2.5.2 Brann og brannspredning

En overflate antennes dersom romtemperaturen i dens umiddelbare nærhet når antenningstemperaturen for overflatens materialtype og at overflaten har vært utsatt for høy temperatur i en gitt tid. Det er i denne sammenheng nødvendig å animere flammer og røyk. Flammene og røyken bør være deler av et strukturert partikkelsystem. Partiklene skal holdes inne i rommet der de har utspringspunkt. Partiklene skal også bli påvirket av luftstrømninger i rommet de befinner seg i. Dersom et punkt på en overflate settes i brann, må temperaturen omkring påvirkes av dette. Brannspredningen skal visualiseres med dette som grunnlag.

2.6 Demonstrasjonsscenarier

Det skal utvikles ett eller flere demonstrasjonsscenarier. Dette er for å ha noe kjørbart å vise i sammenhenger der prosjektet skal presenteres og vurderes. Disse scenariene skal tydeliggjøre verktøyets virkemåte og hensikt. Det er viktig at scenariene er virkelighetsnære, og at brukeren som går igjennom eller får presentert scenariet dermed forstår sammenhengen. Ett av disse scenariene (sykehusscenariet) har et definert hendelsesforløp i spesifikasjonen. Ut over dette står vi fritt til å utvikle demonstrasjonsscenarier etter eget ønske, dersom det blir tid til det. Det vil sannsynligvis også bli aktuelt å sette sammen andre, mindre hendelsesforløp i sykehusmodellen.

2.6.1 Krav til sykehusscenario

Det skal settes sammen et scenario der hendelsene utspiller seg i deler av et sykehusbygg. Modellen av dette bygget må sannsynligvis produseres av en eller flere av gruppemedlemmene. Modellen skal bestå av to rom, fordelt på to etasjer. Bunnetasjen skal inneholde et væskeførende rør i en ende, samt noe telekommunikasjonsutstyr i form av nettverkstjenere, i den andre enden. Etasjene skal dele en ventilasjonssjakt. Denne skal

brukes til å vise hvordan ventilasjonssystemer fungerer som aksellerator for spredning av varme og røyk. Andre etasje skal være et vanlig pasientværelse eller et annet typisk sykehusværelse (lager, møterom, kott e.l.). Hendelsesforløpet i demonstrasjonsscenariet forløper slik:

1. Lekkasje i væskeførende rør igangsettes manuelt.
2. Væskenivå stiger relativt til utstrømingsvolum og rommets dimensjoner.
3. Telekommunikasjonsutstyr kortslutter i det nivået når opp til sårbare komponenter.
4. Tekstlig eller grafisk indikator på svikt i telekommunikasjonsutstyr.
5. Kortslutning i telekommunikasjonsutstyr setter i gang en brann.
6. Katastrofal spredning av brann.

Det er ønskelig at det er tatt høyde for avrenning av væske fra rommet, men dette er ikke et krav.

2.7 Utgivelser

Det vil bli lagt ut en oppdatering på hjemmesiden for prosjektet etter hver fullførte sprint. Hjemmesiden er ment for å gi en generell beskrivelse av bachelorprosjektet, oppgaven, oppdragsgiver, veileder og gruppen. Vi vil legge ut oppdateringer på denne siden underveis i prosjektet i form av videoklipp av den siste implementasjonen. Dette fordi det skal bli lettere for veileder, oppdragsgiver og eventuelt andre interessenter å følge med på prosjektets fremdrift. I tillegg vil veileder og oppdragsgiver bli informert på status- og veiledningsmøtene. Vi vil også levere tre statusrapporter underveis hvor vi skisserer hva vi har fått til og hvordan vi synes arbeidet med prosjektet har gått den siste perioden. Dette vil være et supplement til status- og veiledningsmøtene. Ved behov for ekstra veiledning eller om vi får behov for å klarere uklarheter, vil vi forsøke å løse dette over e-post, skype eller ekstra veiledningsmøte hvis veileder/oppdragsgiver er tilgjengelig for dette.

2.8 Rutiner for organisering og kvalitetssikring

2.8.1 Dokumentasjon

Dokumentering av ny funksjonalitet skal til enhver tid utføres simultant med kodingen. Vi har som en regel innad i gruppen at alt man ikke vil huske om 14 dager skal dokumenteres. Hver gang en modul er ferdig implementert skal det dokumenteres til prosjektrapporten om funksjonaliteten og hvordan det er tenkt underveis i utviklingen.

2.8.2 Backup

Backup av all kildekode og dokumentasjon gjøres hver fredag. Backupen brennes på CD/DVD og oppbevares på en slik måte at den ikke forsvinner eller blir ødelagt.

2.8.3 Konfigurasjonsstyring

Vi benytter oss av ANT til å holde orden på build-rekkefølgen. Dette kan integreres i Eclipse og automatiseres.

2.8.4 Testing

Etter hver sprint i utvikling, hver 14. dag, skal systemet testes. Hver systemmodul testes for å kunne avdekke feil og mangler ved applikasjonen. Dette vil redusere risikoen for å

få ubehagelige overraskelser i slutten av prosjektperioden. Det skal for hver test skrives en testrapport som skal inneholde hvilke deler som ble testet, hvordan det ble testet og hva som ble avdekket av eventuelle feil og mangler.

2.9 Supplementær kravspesifikasjon

2.9.1 Videreutvikling

Prosjektet er i første omgang ment å være et "Proof of Concept". Vi ønsker å vise at det lar seg gjøre å visualisere kritiske hendelser som følge av forstyrrelser i kritisk infrastruktur, ved hjelp av en spillmotor og utviklingsverktøy basert på fri programvare. Det er vanskelig å estimere hvor lang tid det vil ta å utvikle dette verktøyet. Vi kan derfor ikke se bort fra at det må videreutvikles noe i etterkant av levering. Dette stiller høye krav til god dokumentasjon av koden vi skriver, og generell dokumentasjon av de forutsetninger og beslutninger vi gjør underveis.

2.9.2 Brukervennlighet

Det forventes at brukere av applikasjonen har generelt god kunnskap i databehandling. Applikasjonen skal være enkel i bruk med utstrakt bruk av "pek og klikk".

3 Design

3.1 Innledning

I denne delen har vi tatt for oss hvordan vi har tenkt at designen på applikasjonen skal være. Vi begynner med en grov oversikt over designen, for så å bringe dette videre til detaljnivå etter hvert.

3.1.1 Generelle mål

Overordnet mål for designet er å lage et system hvor vi på en best mulig måte klare å gjenskape virkeligheten. Det er derfor lagt vekt på at de forskjellige effektene og de innvirkningene disse har på omgivelsene er mest mulig realistiske.

3.1.2 Kort beskrivelse av systemet

Verktøyet skal brukes til visualisering og analyse av gjensidig avhengighet mellom kritiske infrastrukturer, og de uheldige konsekvensene slike avhengigheter kan ha. Eksempler på dette kan være brann eller store lekkasjer/oversvømmelse.

3.1.3 Bruksområde

Dette systemet er ment som et hjelpeverktøy i forbindelse med analyse av eksempelvis sikkerheten i en bygning ved brann. Det skal gjøre det lettere å forutse hvilke konsekvenser ulike hendelser kan få, slik at det kan iverksettes korrigerende tiltak som kan hindre en eventuell katastrofe.

Systemet vil i første rekke bli brukt som et proof of concept, og det vil bli aktuelt med en senere videreutvikling hvis behovet for et slik system er tilstede. Videreutvikling vil være avhengig av de resultatene vi kommer frem til løpet av denne prosjektperioden.

3.1.4 Rammer

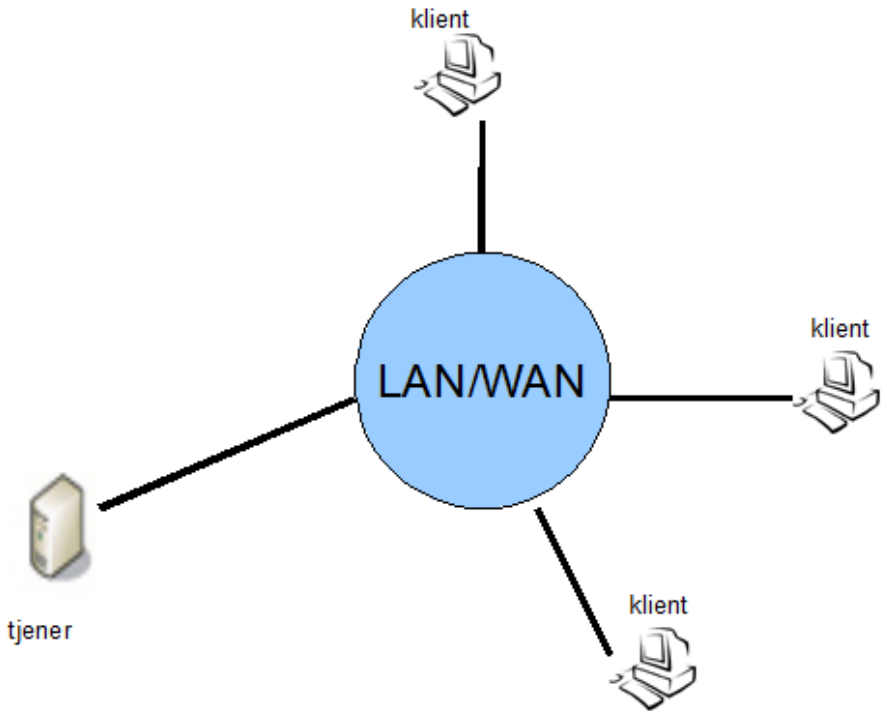
Prosjektet strekker seg over en begrenset tidsperiode, og vi vil underveis i prosjektet bestemme i samarbeid med oppdragsgiver og veileder hvilke elementer som skal prioriteres. Det vil som tidligere nevnt bli lagt stor vekt på realisme i effektene, men ikke estetikk og hvor pene effektene er. Det stilles ingen spesielle krav til maskinvare utenom at det er ønskelig at systemet skal kjøre på en, etter dagens standard, normalutstyrt datamaskin. Dette vil bli vektlagt i all koding og alle valg som blir utført i løpet av utviklingen.

3.2 Overordnet design og systemarkitektur

3.2.1 Beskrivelse av systemet og arkitekturen

Vi vil ved visualisering ikke begrense forflytningen til “spilleren”. Det vil si at det vil være mulig å bevege seg gjennom både vegger og tak, noe som vil gjøre det lettere for brukere av systemet å følge med på det som skjer.

Systemet blir bygget opp rundt en klient/tjener-arkitektur hvor man kan opptre som enten klient eller tjener. Som klient kobler du til en eksisterende server via nettverket og vil kunne se samme scenarie som på tjeneren. Velges derimot å opprette et eget scenarie vil du få tjenerrollen, og andre brukere vil få muligheten til å koble til. Det er serveren som til enhver tid holder på informasjon om begivenhetenes gang. Informasjonen deles



Figur 4: Nettverksarktitekturen for systemet

over nettverket til de klientene som er koblet til.

3.2.2 Klassediagram

Klassediagrammet er en oversikt over de forskjellige klassene systemet består av. Det gjør det lettere å forstå sammenhengen mellom de forskjellige enkeltdelene i systemet, og en bedre oversikt underveis i implementeringen.

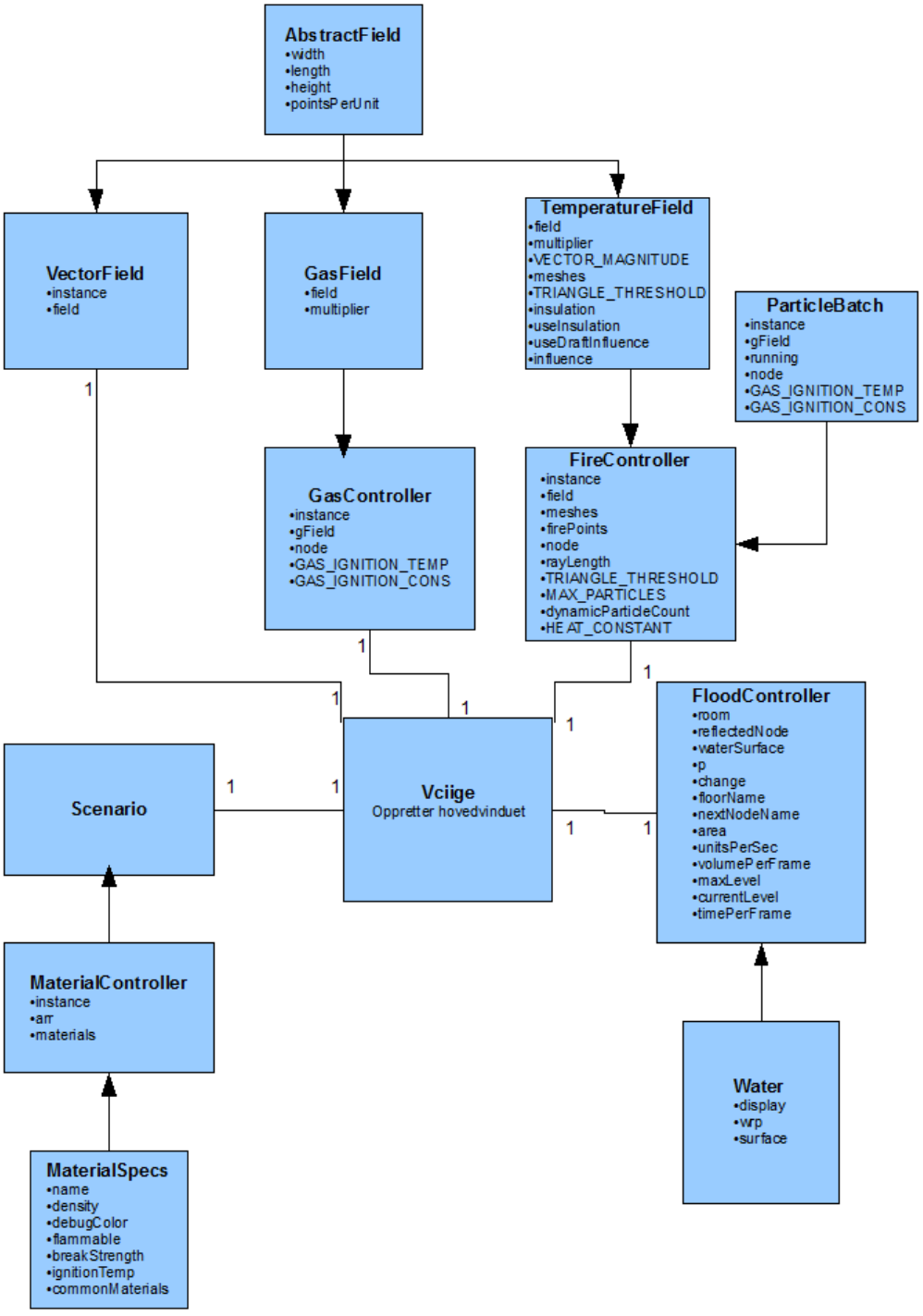
VciigeGamePrototype Dette er hovedklassen som initialiserer all funksjonalitet da systemet starter opp. Det er også via denne klassen oppdateringene av omgivelsene blir kjørt i hver "game loop"/spilloppdatering.

AbstractField Inneholder funksjonalitet og informasjon som er felles for alle feltene i implementasjonen som er diskrete tredimensjonale felt. Holder på informasjon om størrelse på feltet og avstanden mellom punktene.

VectorField Inneholder informasjon om trekk i et rom ved hjelp av vektoren. Disse vektorene blir påvirket av eksempelvis brann, men man kan også gå inn å sette hvor stor trekk man vil ha i et punkt manuelt.

GasField Hvert punkt inneholder informasjon om hvor stor konsentrasjonen av gass er i det aktuelle punktet. Punktene oppdateres i hver "game loop"/spilloppdatering, og gassen spres til nærliggende punkter.

TemperatureField Hvert punkt initialiseres til 20°C ved programstart (i koden opereres det med Kelvin). Alle punktene har innvirkning på sine nabopunkter. Hvis temper-



Figur 5: Konseptuelt klassediagram for implementasjonen

aturen endres i et punkt vil denne gi et bidrag til sine nabopunkter. Oppdaterer hvert punkt i temperaturfeltet for hver spilloppdatering.

GasController Undersøker om konsentrasjonen og temperaturen i noen av punktene i gassfeltet er over de kritiske verdiene. Hvis dette inntreffer vil det opprettes en eksplosjon i det punktet.

FireController Hvis temperaturen er høyere enn kritisk verdi for materialet og materialet har vært utsatt for høy temperatur over lenger tid, opprettes det en brann. Sørger for å påvirke temperaturpunktene rundt oppståtte brannpunkter.

ParticleBatch Klasse benyttet til å opprette eksplosjoner, brann og røyk, samt tilpasning av disse effektene i forhold til rommet de er plassert i.

FloodController Denne klassen sørger for å opprette en vannflate i en bygning. Den oppretter en flate og legger på vanneffekt på denne. Styrer også heving og senking av vann-nivået.

Water Legger på vanneffekt på flaten opprettet i FloodController.

MaterialController Denne klassen holder på informasjon om alle elementer i en bygning, og hvilke materialer hvert element har.

MaterialSpecs Inneholder informasjon om forskjellige typer materialer. Mulighet for å opprette egne materialtyper hvis det er behov for det.

3.3 Design av brukergrensesnitt

Brukeren vil få opp hovedmenyen ved oppstart av programmet hvor han/hun får presentert tilgjengelige valgmuligheter. (figur 6).

- Opprette server
- Koble til server som klient
- Starte “enspiller”-modus
- Endre “spillernavn”
- Avslutt

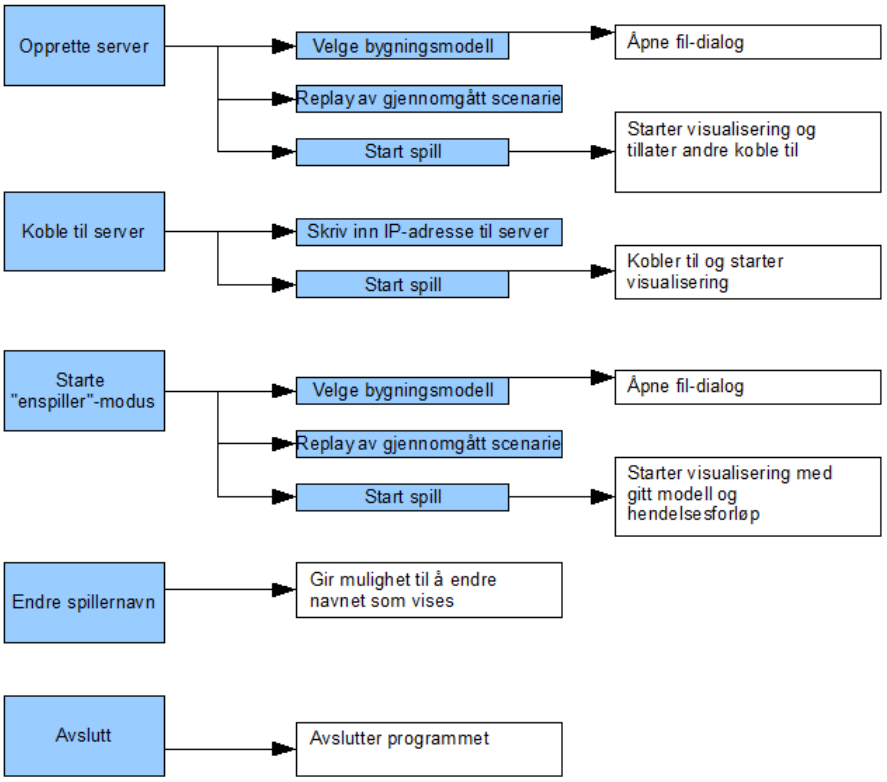
Figur 7 viser hovedmenyen slik det er tenkt at den skal se ut. Dialogene for valg av filer kommer til å bestå av rene ferdiglagede swing-komponenter.

3.4 Design av modeller, effekter og scenarie

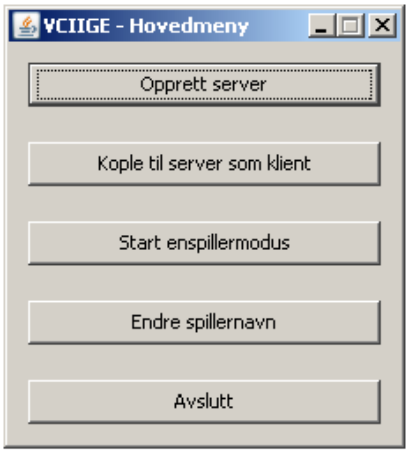
3.4.1 Modeller

I første omgang vil vi benytte oss av et hardkodet rom. På denne måten kan vi raskt komme i gang med implementasjon og testing. Vi vil lage noen egne modeller ved hjelp av Blender som vi senere vil laste inn i spillmotoren. Disse modellene vil danne grunnlaget for scenariene vi skal utvikle mot slutten av prosjektperioden.

Ideelt sett hadde vi her importert en reell modell av en bygningsstruktur, men dette har vi ikke klart å oppdrive. Det finnes mange modeller som er fritt tilgjengelig på internett, men dette er som regel mindre “rekvisitter” og ikke detaljerte modeller av reelle bygninger. Det er naturlig at de som måtte ha CAD-modeller av slike kritiske bygninger



Figur 6: Navigasjonskart over hovedmenyen



Figur 7: Forslag til layout i hovedmeny

ikke gjør disse tilgjengelige. Det er uansett mulig å importere dette til Blender da det her er et innebygget importeringsskript for dxf-filer, eventuelt kan modellen lastes direkte inn i MonkeyWorld3D hvis det er et annet støttet format. Her kan modeller legges inn i nodestrukturen, og på den måten blir modellen klargjort til innlasting til spillmotoren.

3.4.2 Effekter

jMonkeyEngine har støtte for flesteparten av de effektene vi får behov for: ild, røyk, eksplosjon, vann. Det finnes et velfungerende system for å legge vannoverflate på objekter som vi spesialtilpasser til vårt formål. Det samme gjelder partikkeleffektene ild, røyk og eksplosjon. Her bruker vi partikkel-editoren til å lage våre egne effekter, eventuelt går det an å bruke ferdigdefinerte effekter som ligger lagret i det nevnte verktøyet.

4 Implementering, koding og produksjon

Vi har benyttet oss av Eclipse Java IDE til kodeskrivingen i prosjektet. Eclipse har støtte for SVN ved installasjon av en plugin. Dette gjorde at vi kunne hente inn alle prosjektene vi trengte i prosjektet, jME, JGN og jMEPhysics, direkte inn i Eclipse, samt at vi hadde full kontroll over vårt eget prosjekt med versjonshåndtering. Eclipse viser automatisk frem JavaDoc for en klasse eller funksjon, noe som gjorde det lettere for oss å finne frem til funksjonalitet og forstå denne underveis i kodingen.

Vi har under hele perioden bestrebet oss på å holde en god struktur på organisering av kode og dokumentasjon av kode. Det vi legger i begrepet god struktur er at vi holder oss til samme innrykksregler overalt i koden og god dokumentasjon av egenprodusert kode. Dokumentasjon av kode følger retningslinjene for JavaDoc, men elementer i koden som kun er for intern hjelp er kommentert som vanlige programlinjekommentarer.

4.1 Strukturen i hovedklassen

Forklaring på hovedklassen *VciigeGamePrototype*. Det er denne klassen som initialiserer og setter i gang alt, det er også denne som kontinuerlig driver spillet videre helt til brukeren trykker escape. Grovt sett kan den deles opp i tre:

```
public class VciigeGamePrototype extends SimplePassGame {

   // Medlemsvariable
   private GameObject gameObject;

   public static void main (String[] args) {
      VciigeGamePrototype game = new VciigeGamePrototype();
      // Følgende starter det hele:
      game.start();
   }

   protected void simpleUpdate () {
      // Denne funksjonen blir kjørt en gang for hver gang
      // bildet tegnes opp på nytt. Det er denne som driver
      // spillet videre.

      // Et veldig viktig moment er timeren. Denne kalles for
      // hver oppdatering og påvirker objektene som skal
      // oppdateres. Variabelen time vil inneholde tiden brukt
      // på forrige oppdatering denne kan variere en god del
      // avhengig av hvor tungt spillet kjører
      float time = timer.getTimePerFrame();

      // Objekter som er relevante for spillets framdrift
      // får et kall på sin update-metode og oppdaterer så
      // med påvirkningen av den nevnte tidsfaktoren.
      gameObject.update(time);

   }

   protected void simpleInitGame () {
      // Her opprettes og instansieres alt som trengs før
      // oppdateringene begynner og spillet drives i gang.
```

```
    }

}
```

4.2 Materialsystem

Materialsystemet definerer hvilke materialer et romlig legeme består av, for eksempel vegg, tak eller gulv. Spillmotoren har innebygde materialer definert i fysikksystemet, men vi hadde behov for å tilegne materialer andre egenskaper enn det som var definert der. Vi endte derfor opp med å lage en egen klasse, *MaterialSpecs*, som har mye av den samme funksjonaliteten som den innebygde materialklassen; men mer utvidet informasjon. Denne inneholder noen statiske ferdigdefinerte materialer vi mener det er mest sannsynlig at vi vil få bruk for. Det er i tillegg mulig å opprette nye materialer hvis det skulle være behov for materialtyper utover de ferdigdefinerte. Materialene har variable som indikerer hvor brennbart det er og hvor mye trykk det tåler før det gir etter. Mest interessant for vårt tilfelle til å begynne med er brennbarheten til materialene, men trykk vil også være relevant for eventuell videreutvikling. Det vil være mulig å utvide denne klassen med funksjonalitet hvis det på et senere tidspunkt blir behov for det.

Materialene som er tilegnet spatial-elementene styres av klassen *MaterialController*. Dette er en singletonklasse, en klasse med bare én instans, som har to *ArrayList*-objekter.

```
    //material types
    private ArrayList<MaterialSpecs> arr;
    //Holds spatial and its material
    private ArrayList<Vector<Spatial>> materials;
```

Arraylisten 'arr' inneholder alle de ferdigdefinerte materialene i systemet, og arraylisten 'materials' inneholder vectorer med spatialobjekter, vegger, tak og lignende. Indeksene i de to arrayene har en virtuell link mellom seg, slik at elementene som ligger med indeks 1 i 'materials' har den materialtypen med indeks 1 i 'arr'.

4.3 Temperatur

I et rom har vi tenkt oss at luften er delt opp i biter/klosser slik at all luft omsluttet av denne tenkte klossen har samme temperatur. Videre har vi tenkt at varmekapasiteten til en slik kloss er lik alle de andre. Dette innebærer at når en bit mister varmeenergi tilsvarende en temperaturendring på f.eks. 1 Kelvin så vil naboklossen(e) ta til seg denne energien og få en økning som tilsvarer akkurat 1 Kelvin. Et såkalt temperaturfelt er en samling punkter i rommet (figur 8) representert av en tredimensjonal array av flyttall. Hvert flyttall representerer temperaturen i det romlige punktet. Dette punktet har vi tenkt oss at befinner seg i midten av klossen.

Når tiden går, vil temperaturen/varmen gradvis spre seg rundt i rommet. Det vil si at hvis det ikke er noen varmekilder i systemet, vil temperaturen i alle punktene nærme seg deres gjennomsnittstemperatur. Hvis det oppstår en varmekilde (brann) i rommet, vil alle punktene bli gradvis varmere da det ikke er tatt med tap av varme til omgivelsene som befinner seg utenfor rommet.

I en spilloppdatering/"game loop" blir alle slike punkt i temperaturfeltet oppdatert én gang. I en oppdatering av et punkt blir det utvekslet varme med dets nærmeste nabopunkt. Det vil si 6 punkter totalt (positiv/negativ x-, y- og z-retning).

Figur 8: En enkel visualisering av datastrukturen.

4.3.1 Vektorfelt

Her er det også tatt høyde for at det kan være luftstrømninger i et rom som påvirker graden av varmeutveksling mellom nabopunkter (figur 9). Dette er løst ved å bruke et vektorfelt som bygger mye på samme konseptet som temperaturfeltet. Også her blir rommet oppdelt i en mengde punkter som har en vektorstørrelse tilordnet seg i stedet for et flyttall. Denne har en retning og en lengde som representerer henholdsvis luftstrømmens retning og styrke i akkurat dette punktet. Når man bruker disse feltene sammen er de like store og like finmasket slik at oppdelingene blir sammenfallende.

Figur 9: Her vil det i svært liten grad bli varmeutveksling mot venstre sammenlignet høyre retning. Sirklene illustrerer temperaturpunkt, pilene representerer retningen og styrken på lufttrekken i punktet.

4.3.2 Isoleringseffekt

Dette simuleres ved å finne ut hvilke punkter i temperaturfeltet som har en vegg eller noe annet mellom seg. Alle disse parene blir lagret i en datastruktur for senere bruk. Når varmeoverføringen senere går sin gang vil punktene parvis bli sammenlignet med de i datastrukturen. Får man et treff her, blir det ikke utvekslet varme mellom punktene. Siden denne datastrukturen bare fylles én gang, tar vi ikke høyde for at omgivelsene i rommet kan endre seg. Skulle man tatt høyde for dette, har vi erfart at det ville blitt for mange utregninger til at det kunne kjøre i sanntid. Da måtte man sjekke alle trianglene i scenen for hvert eneste par med punkter som skulle utveksle varme.

Her er det ikke noe problem å skille mellom delvis og fullstendig isolering. Verdier for isoleringsgrad kan tilegnes forskjellige materialer. Vår løsning bruker ikke isolering

aktivt da dette er en stor ekstra belastning, men koden for det er laget.

4.4 Brann

Klassen *FireController* tar hånd om funksjonaliteten som omhandler branntilløp og brannspredning. Det er en singleton-klasse som gjør den lett tilgjengelig på tvers av klasser.

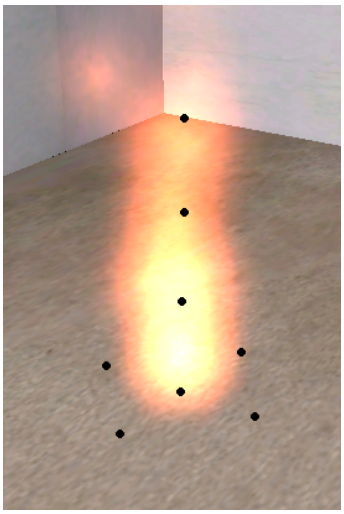
Når programmet startes, blir alle grupperinger av bygningsdeler gjennomgått. De brennbare skilles fra de som ikke er brennbare og blir tatt vare på i en datastruktur. Et annet krav er at de må være under en viss geometrisk kompleksitet, dvs. bestå av mindre enn et gitt antall triangler. Ellers blir videre behandling så treg at bilderaten blir uholdbar.

Temperaturfeltet er en diskret fordeling av punkter i rommet som inneholder informasjon om temperaturen i rommet. På objekter som kan ta fyr blir det plassert ut punkter som vi har kalt for *FirePoint*s. Jo kortere avstand det er mellom temperaturpunktene i feltet, jo kortere avstand er det mellom disse brannpunktene. Det er kun i disse punktene at en flamme kan oppstå. Disse punktene akkumulerer "varmeenergi," noe vi har definert som temperatur multiplisert med tid. For at det skal ta fyr i et punkt, må det ha blitt akkumulert en viss varmeenergi. I tillegg må temperaturen i det nærmeste temperaturpunktet være høy nok. Følgende kode blir kjørt hver gang bildet tegnes opp (blir kalt på av *simpleUpdate()* i hovedklassen):

```
public void update(float interpolation) {

   // Går gjennom alle brannpunkter i scenen

   for (int i = 0, max = firePoints.size(); i < max; i++) {

      FirePoint fp = firePoints.get(i);
      // Dette er det nærmeste punktet i temperaturfeltet
      Vector3f loc = fp.getClosestPoint();
      // Posisjonen til brannpunktet
      Vector3f source = fp.getPosition();
      // Branntemperaturen til dette brannpunktet (avhenger av
      // hva slags materiale som ligger til grunn).
      float temp = fp.getCombTemp();

      // Hvis temperaturen like ved er minst 80% av
      // branntemperaturen, akkumuleres det varmeenergi.
      if (field.getTemp(loc) > temp * 0.8f) {
         fp.addHeat(field.getTemp(loc) * interpolation);
      }

      // Det tar fyr i punktet hvis følgende krav er oppfylt:
      // - Det har ikke allerede tatt fyr
      // - Det er akkumulert tilstrekkelig varmeenergi
      // - Temperaturen i lufta like ved er minst 85%
      // av branntemperaturen (materialavhengig).
      if (fp.getHeat() >= (HEAT_CONSTANT * temp) && !fp.isInFlames()
            && field.getTemp(loc) >= (temp * 0.85f)) {
         setFire(source);
      }
   }
}
```

Her har vi kommentert koden ekstra nøye og på norsk for eksemplets skyld. I tillegg til denne metoden er det en til som blir kjørt for hver oppdatering: Metoden som gjør at allerede oppståtte branner varmer opp lufta rundt seg. For hvert *FirePoint* som har tatt

fyr settes temperaturen i nabopunktene til en prosentandel av forbrenningstemperaturen (denne avhenger av materiale). Dette skjer bare hvis det medfører en temperaturøkning, slik at flammen ikke kan riskikere å få en avkjølende effekt. Dette fører altså til at det blir tilført varme til systemet, gjennomsnittstemperaturen øker og brannen sprer seg på brennbare materialer.



Figur 10: Disse punktene får økt temperatur av flammen.

4.5 Soteffekt

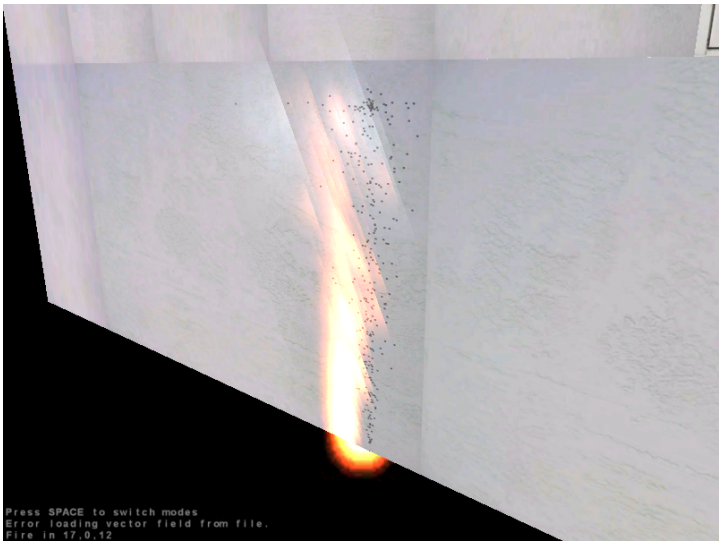
Når flammer eller røyk kommer i kontakt med overflater, vil det dannes et sotlag. Denne effekten var det ønskelig at vi fikk med i visualiseringen. Motivasjonen for å kunne visualisere sot er at sotlaget er karbonholdig og at det derfor kan lede strøm. Hvis for eksempel røyk blir sugd inn i en server eller en annen datamaskin, kan dette kortslutte maskinen og være potensiell årsak til kollaps i en kritisk infrastruktur.

For å visualisere sot kom vi fram til en teknikk som kalles “texture splatting”. Det tok lang tid å finne ut av hvordan vi skulle løse dette, og her henvendte vi oss mye til forumet for ledetråder. Det endte opp med at vi bygde videre på et eksempel laget av en anonym bidragsyter til jMonkeyEngine med kallenavn “hevee”. Hans eksempel går ut på å bruke et bildebuffer til å “tegne” på for så å oppdatere teksturene. Disse teksturene blir så kombinert og sluttresultatet ser ut som om det har blitt tegnet på baseteksturen. For alle flater som skal kunne bli påvirket av denne effekten må det lages slike bufrede bilder og ekstra teksturlag.

En liten prosentandel av ild- og røykpartiklene “tegner” på grå prikker på det ene teksturlaget hvis de kommer i kontakt med noen av disse flatene (figur 11). Denne teksturen blir så oppdatert og lagt på flaten på nytt. Disse teksturene kombineres på en slik måte at veggteksturen synes gjennom teksturen som representerer sotlaget [22].

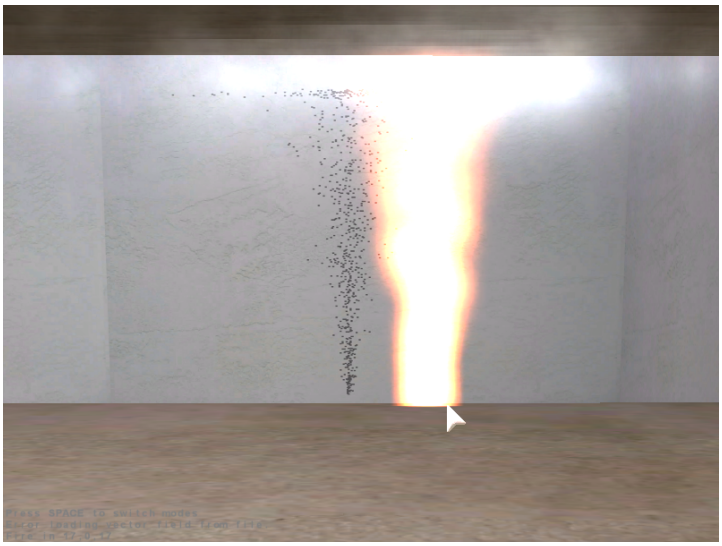
Denne måten å kombinere to teksturer på krever at pc-ens skjermkort har støtte for minst to teksturenheter. Dette er ikke noe problem selv for grafikkort av eldre type. En av våre arbeidsstasjoner har et ATI Radeon X800 Pro (dette er et gammelt kort) og det har støtte for 16 slike teksturenheter.

Denne effekten er i et tidlig stadie og er ikke stabil nok til å kjøre sammen med alt



Figur 11: Slik ser soteffekten ut på baksiden av veggen. Det er lettere å se effekten fra denne vinkelen.

annet av effekter. Etter noen sekunder krasjer spillet og man får beskjed om at det har gått tomt for minne. Dette har vi ikke hatt tid til å finne en løsning på, da soteffekten er noe av det siste vi har holdt på med i dette prosjektet. Et annet problem med denne effekten er at man får en sideforskyvning av soten (figur 12). Årsaken til denne feilen er heller ikke funnet.



Figur 12: Sideforskyvning av sot. Denne øker desto lengre ut til siden man kommer på flaten.

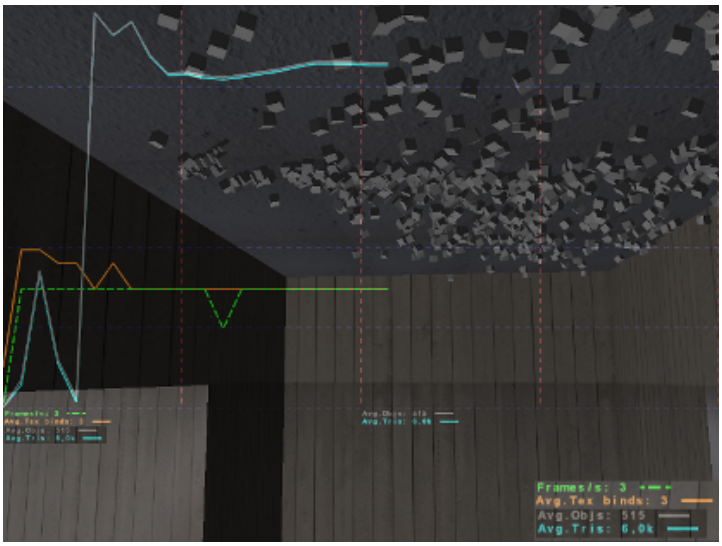
4.6 Partikkelsystem

Utviklingsprosessen har ikke vært fri for problemstillinger, disse har ofte blitt løst ved at vi har justert ulike konsepter i spillmotoren til å passe våre behov. Vi har lært at det å

sette sammen et simuleringsverktøy er noe ganske annet enn å utvikle et spill. Effekter og påvirkninger man ofte ville ignorert i spillsammenheng, er uunnværlig i simuleringssammenhenger.

En av de større problemstillingene vi har støtt på, har vært å skape påvirkninger på effekter som brann, røyk osv. Vår veileder har hele tiden ytret et ønske om at vi prioriterer realisme. Vi har så langt det har latt seg gjøre forsøkt å oppfylle dette. Det viktigste her er brann- og røykeffekter. For å få på plass dette begynte vi å studere fysikkmotoren jME Physics 2.0, som er et separat fysikktillegg til jMonkeyEngine. Idéen var å lage et dedikert partikkelsystem som benyttet seg av denne. Denne fremgangsmåten hadde slik vi så det fordeler med tanke på realisme i forhold til partikkelbevegelse og kollisjon, men ulemper i forhold til estetikk og ytelse. Siden vi ønsket et virkelighetsnært visuelt preg og god ytelse valgte vi å jobbe videre med dette.

Underveis i utviklingen av dette partikkelsystemet møtte vi på et problem; systemet greide ikke å kalkulere alle de fysiske påvirkningene på partiklene i sanntid med en akseptabel bilderate. For å få generert noe som likner en brann med tilhørende røyk måtte vi ha titusenvis av partikler. Etter flere forsøk med optimalisering kom vi fram til at denne løsningen ble for kostbar ytelsesmessig. Under testing med 1500 simplistiske partikler oppnådde vi tre til fem bilder i sekundet på vår kraftigste datamaskin. Figur 13 viser en slik testkjøring med oppskalerte partikler med minimal geometrisk kompleksitet. Dette er langt unna akseptabel ytelse. Vi har derfor valgt å tilpasse det langt lettere eksis-



Figur 13: Ytelsestest av partikkelsystem med jME Physics 2.0

terende partikkelsystemet i spillmotoren. Dette partikkelsystemet fungerer slik at vi kan definere utstrømningspunkter og plassere disse i det tredimensjonale rommet. Hvordan partiklene ser ut og hvordan de strømmer ut i fra utstrømningspunktet, kan reguleres og tilpasses. Det finnes ferdiglagede partikkelpåvirkninger i spillmotoren. Vi har tilpasset noen av disse, men vi har også hatt behov for å produsere egne påvirkninger. Ved å benytte det eksisterende partikkelsystemet har vi spart mye tid. En del funksjonalitet er ferdiglaget, og det visuelle er enklere å få på plass. På den andre siden er det ikke like enkelt å registrere partikkelkollisjon og overflateberøring.

Vi har satt sammen en partikkelpåvirkning som registrerer om en partikkel berører en overflate (vegg/tak/gulv) eller om den har passert denne overflaten. På denne måten kan vi, ved å kalkulere ny partikkelbevegelse og hastighet, tilnærme oppførselen vi ville oppnådd ved å benytte oss av fysikkmotoren. Visuelt og ytelsesmessig sett fungerer dette veldig bra. Vi kan ha tapt noe på realisme, men vi har et begrenset tidsvindu å forholde oss til. Med det i tankene mener vi å ha funnet en grei løsning på problemstillingen. Figur 14 viser en flamme med røyk, generert vha det eksisterende partikkelsystemet, og med bruk av vår egenproduserte overflatepåvirkning. Partiklene blir alle tilegnet en

Figur 14: Brann og røyk med overflatepåvirkning

bildetekstur som fargelegges avhenging av hva den skal representere. Teksturen følger dimensjonene på partiklene som gjerne forandrer seg under partiklenes livsløp. For å få til gjennomsiktighet på disse teksturene benytter vi spillmotorens innebygde metoder for alpha-blending.

4.6.1 Oppbygging av partikkelsamlinger

ParticleBatch-klassen ble laget for å gjøre det enkelt å håndtere partikkelsamlinger. I utgangspunktet var det ønskelig for oss å kunne laste inn partikkelmodeller generert av partikkelsystem-editoren som er tilgjengelig i spillmotoren. For å få til dette har vi benyttet oss av en ferdiglaget binærfil-importør i spillmotoren. Det er også mulig å hard-kode partikkelsamlinger, eksempler på hvordan dette gjøres er dokumentert i *Particle-Batch*-klassen. Selv om partikkelsamlinger lastes inn fra binærfil, er det fremdeles mulig å endre partikkelsamlingens generelle visuelle egenskaper (størrelse, partikkellevetid, ytre påvirkning osv.). Vi kan også endre partiklenes bevegelse ved hjelp av forskjellige påvirkninger. Dette er klasser som er bygd opp etter en viss struktur som påvirker oppførselen til partiklene. For å holde partiklene som genereres innenfor bygningens begrensende flater (vegger, tak, gulv), har vi benyttes oss av en egenutviklet påvirkning som er døpt *SurfaceInfluence*.

4.6.2 Overflatepåvirkning av partikler

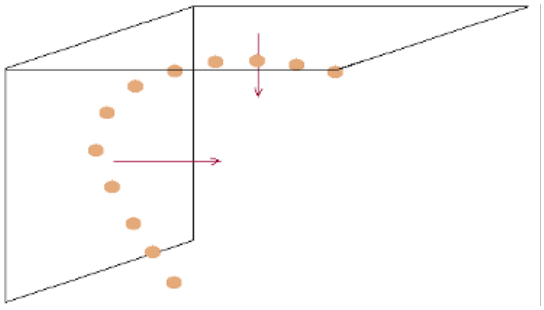
I spillmotorens partikkelsystem finnes ingen ferdiglagde funksjoner for å begrense rommet partiklene beveger seg i. For å kunne visualisere en brann på en noenlunde realistisk måte, har vi måttet implementere denne funksjonaliteten selv. Vi har allikevel fått noen idéer fra spillmotor-biblioteket om hvordan vi skulle gå frem for å løse denne problemstillingen. I spillmotor-biblioteket finnes det en klasse kalt *FloorInfluence*. Denne klassen er ment brukt for å stoppe fallende partikler som treffer et gulv.

Klassen *SurfaceInfluence* fungerer slik at den instansierer et matematisk plan (*Plane*) for en gitt overflate, basert på en posisjonsvektor og en normalvektor. *SurfaceInfluence* inneholder en metode som oppdateres for hvert bilde som blir generert under kjøring. Dersom en partikkel befinner seg på negativ side i forhold til planets definerte normal, vil den bli reflektert med en passende hastighet og vinkel. Dette er selvfølgelig en tilnærming til virkeligheten. Følgende kode blir kjørt i *SurfaceInfluence* for hvert bilde som tegnes:

```
public void apply(float dt, Particle particle, int index) {
   Vector3f newV;
   // Dersom partikkelen ligger på, eller har passert planet.
   if(surface.pseudoDistance(particle.getPosition().divide(SCALE)) <= 0) {
      // Dersom planet representerer en vegg, eller et gulv.
      if(normal.y != -1) {
         // Reflekter partikkelen av overflaten.
         newV = particle.getVelocity().add(normal.mult(SIDE_REFLECT));
         particle.setVelocity(newV);
         // Tilfeldig splatting av sot.
         if(Math.random() < SPLAT_PERCENTAGE) {
            SootSplatter.getSootSplatter().splat(
                  particle.getPosition().divide(SCALE));
         }
      // Dersom planet representerer et tak.
      } else {
         particle.getVelocity().setY(DWN_REFLECT);
         // Tilfeldig splatting av sot.
         if(Math.random() < SPLAT_PERCENTAGE) {
            SootSplatter.getSootSplatter().splat(
                  particle.getPosition().divide(SCALE));
         }
      }
   }
   // Dirty fix. Laget for å unngå at partikler blir synlige i andre etasjer.
   // Dersom planet representerer et tak
   else if (normal.y == -1) {
      // Dersom partikkelen nærmer seg takoverflaten.
      if (closeToRoof(particle)) {
         // Reflekter partikkelen forsiktig ned.
         particle.getVelocity().setY(DWN_REFLECT);
      }
      // Tilfeldig splatting av sot.
      if(Math.random() < SPLAT_PERCENTAGE) {
         SootSplatter.getSootSplatter().splat(
               particle.getPosition().divide(SCALE));
      }
   }
}
```

Det er viktig å merke seg at denne påvirkningen må legges til for hver eneste overflate i rommet. Dette innebærer at det (for et vanlig fire-veggers rom) må legges til seks overflatepåvirkninger (inkl. vegg og tak) for hver *ParticleBatch* (eksempelvis brannpunkt). *SurfaceInfluence* behandler partiklene forskjellig avhengig av hvilken type overflate par-

tikkelen kolliderer med. Dette er fordi vi ønsker oss en mer horisontal fordeling av røyk- og brannpartikler dersom de skulle treffe et takplan, enn om de skulle treffe en vegg. Figur 15 viser en prinsippskisse av virkemåten. Pilene indikerer planenes normaler. Det



Figur 15: Prinsippskisse, overflatepåvirkning.

har vært en utfordring å finne normaler på alle plan som utgjør et rom, spesiellt når det gjelder vegger. Dette er ikke nødvendigvis korrekt definert etter innlasting og eventuell konvertering av en bygningsmodell. Vi har løst dette i *ParticleBatch* ved å avfyre stråler i samme retning som den fordefinerte normalen. Disse registerer treff med andre romlige legemer i samme node. Ved bomskudd inverteres den fordefinerte normalen. Dersom en modell er ryddig strukturert, med romnoder og beskrivende identifikatorer, vil overflatepåvirkning bli satt uten problemer.

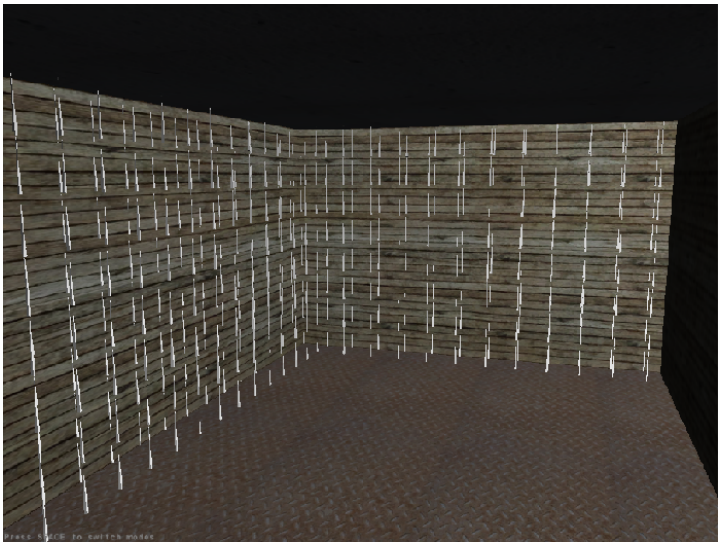
4.6.3 Partikkelpåvirkning av vektorfelt

En brann vil påvirke luftstrømmen i et rom slik at luftstrømmen rundt en brann vil bevege seg mer i samme retning som brannen, men samtidig litt ut til siden for den. Dette vil føre til at luftstrømmene rundt brannen vil peke skrått oppover til høyre eller venstre avhengig av plasseringen i forhold til brannen. Partiklene vil ikke utgjøre et stort bidrag til endringen på luftstrømmen med en gang, men over tid.

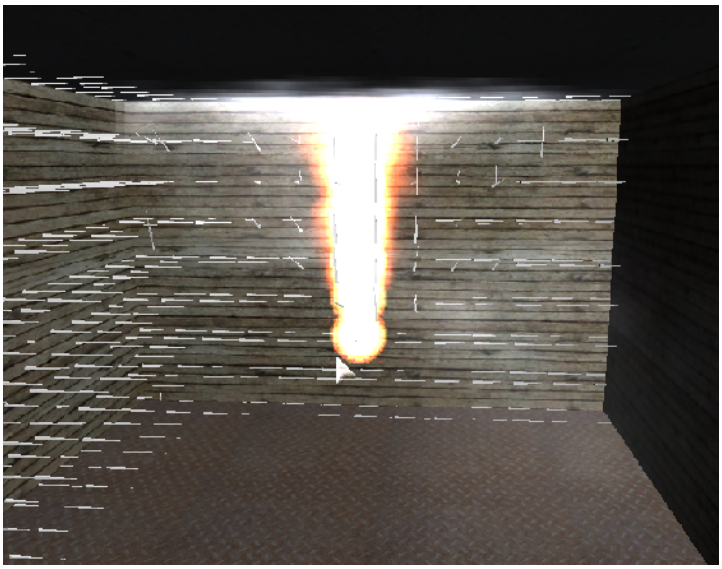
Den beste tilnærmingen til partikkelpåvirkning ville være å benytte fysikksystemet som er innebygget i spillmotoren. Vi gikk tidligere bort fra dette fysikksystemet som en følge av ytelsesproblemer i forbindelse med brann. Utregningene tilhørende partikkelpåvirkning av vektorfeltet ville bli enormt kompliserte hvis vi skulle oppfylle kravet til et dynamisk vektorfelt. Disse utregningene kommer inn under fagområdet termofysikk, og ligger et godt stykke utenfor vårt kompetanseområde. Vi har derfor valgt å la et tilfeldig utvalg av partikler i hvert enkelt partikkelutspring påvirke vektorene i umiddelbar nærhet til det aktuelle partikkelutspringet.

Dersom det eksisterer ett eller flere partikkelutspring i scenen, vil vektorene rundt disse oppdateres for hver spilloppdatering/“game loop”. For hvert partikkelutspring velges det ut 10% av partiklene som skal påvirke vektorene. Disse velges ut ved å generere et tilfeldig tall mellom 0 og antall partikler i partikkelutspringet. Den eksisterende vektoren i det nærmeste punktet til partikkelen blir interpolert mot partikkelens hastighetsvektor. Dette fører til at vektorene rundt partikkelutspringet vil peke mer og mer mot partiklenes retning. Langs alle veggene ble det satt ut piler for for å illustrere hvordan vektorene forandrer seg. I utgangspunktet peker alle pilene i samme retning (figur 16). Ved utsettelse av en brann vil disse pilene endre retning etter hvert som de blir påvirket av

partiklene (figur 17).[1]



Figur 16: Enkel visualisering av vektorfeltet ved spillstart - ingen partikkelpåvirkning.



Figur 17: En enkel visualisering av vektorfeltet med partikkelpåvirkning.

4.6.4 Vektorfeltpåvirkning av partikler

Vektorfeltet blir som nevnt tidligere påvirket av brannpartiklene. Vektorfeltet vil også påvirke partiklenes retning og hastighet. Vi har kun konfigurert vektorfeltet til å påvirke røykpartiklene, da det er røyk som vil være mest følsom for trekk. Brannpartiklene blir ikke påvirket direkte av vektorfeltet grunnet problemer med ytelse ved visualisering av

[1] Disse pilene er kun ment for illustrasjon av hvordan vektorfeltet påvirkes, og vil ikke være en del av den ferdige implementasjonen.

dette. I tillegg vil vi få et bidrag fra partiklene på vektorfeltet og vice versa som vil gjøre at flammen vil gjøre små utfall til hver side for utspringet og tilbake igjen. Vektorfeltet vil derimot påvirke spredningen av brannen, slik at brannen vil spre seg i samme retning som trekk-vektorene.

Røykpartiklenes hastighetsvektor oppdateres for hver spilloppdatering/"game loop". Røyken vil bevege seg i samme retning som vinden, samtidig som partiklenes hastighet øker.

4.7 Gass-systemet

4.7.1 Gassfelt

Gassfeltet er bygget opp rundt samme prinsipp som temperaturfelt med mange punkter fordelt rundt i hele rommet. Hvert enkelt punkt inneholder informasjon om hvor stor konsentrasjonen av gass er akkurat der. Vi har tatt utgangspunkt i gassen propan, C_3H_8, da vi skulle bestemme ved hvilken temperatur gassen skal antenne og hvilket intervall konsentrasjonen må ligge i for å kunne antenne. Det er mange typer gasser som kan være aktuelle i forbindelse med industriell produksjon, men vi antar at propan er den gassen som forekommer hyppigst i og med at denne er lett tilgjengelig.

Propan er som kjent tyngre enn luft, og spredningstendensen vil derfor være nedover mot gulvplanet. Vi har forsøkt å implementere dette så reellt som mulig; slik at det vil spres mer til underliggende nabopunkter enn sidepunktene.

For at propan skal antenne i luft er det nødvendig at konsentrasjonen ligger mellom 2.1 og 9.5% og at temperaturen er større enn 723 Kelvin [2]. I oksygen øker den øvre grensen til 48%.[16]

Hvis konsentrasjonen av gass overstiger 9.5% i luft vil det ikke være tilstrekkelig oksygen tilstede for at en brann/eksplosjon skal kunne oppstå.

4.7.2 Gasskontroller

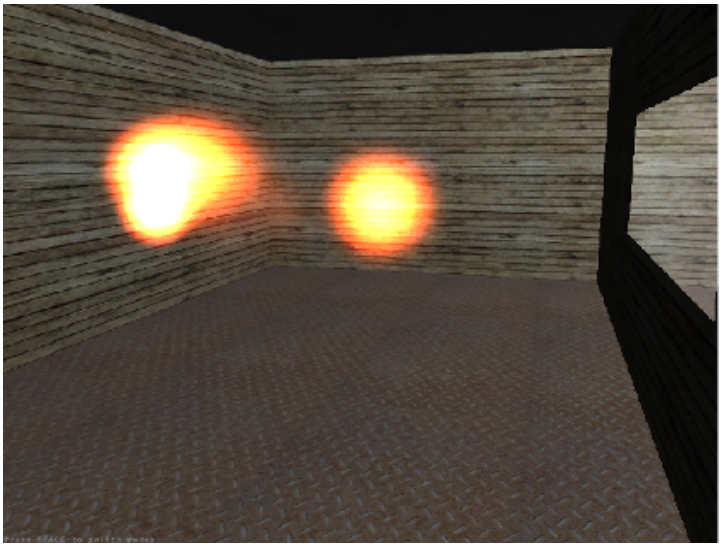
GasController-klassen er en singletonklasse som for hver spilloppdatering går gjennom og sjekker alle punktene i gassfeltet. Den henter også ut temperaturen i korresponderende punkt i temperaturfeltet. Hvis temperaturen er lik eller høyere enn antennelsestemperaturen og konsentrasjonen av gass i punktet ligger i antennelsesintervallet, vil det ved hjelp av FireControlleren settes i gang en brann/eksplosjon i de aktuelle punktene.

Visualiseringen av eksplosjonene er ikke helt optimal i forhold til det virkelige liv. Vi har forsøkt å lage en algoritme som prøver å spre eksplosjonene til de nabopunktene som har størst konsentrasjon av gass. Det var ønskelig å få til en tilnærming slik at eksplosjonene endte der hvor gasslekkasjen startet. Funksjonen gjorde at ekslosjonene spredte seg mer tilfeldig ut fra utgangspunktet, men allikevel viste det seg å være vanskelig å tilnærme dette til virkeligheten. Eksempel på hvordan eksplosjonene spredte seg fremgår av figur 18.

4.8 Oversvømmelse

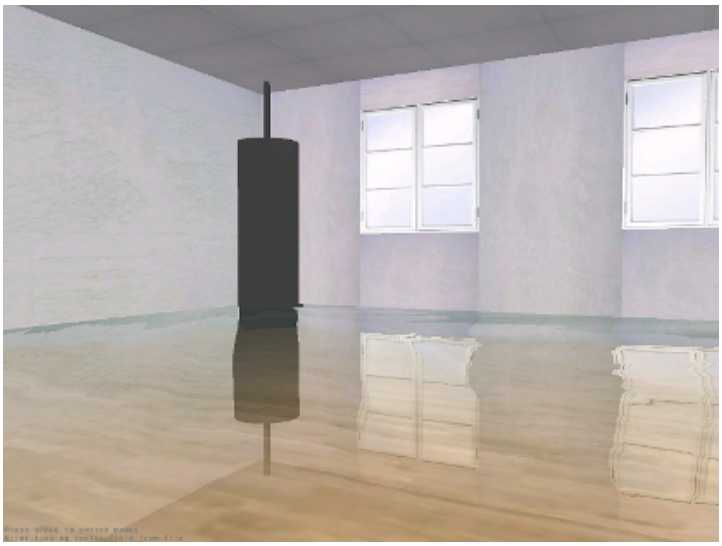
Visualisering av oversvømmelse var noe av det første vi begynte å jobbe med, under implementeringen. Oversvømmelser visualiseres som vannoverflater vha. *Water*-klassen. Disse er bygget opp som enkle plan, og har ingen utstrekning/masse. Overflatene er bygd opp av små polygonnett (*TriMesh*), som oftest med to polygoner. Polygonnettet blir

[2] Tilsvarer 450°C



Figur 18: Første tilnærming av eksplosjoner i forbindelse med gass-spredning

overtrukket med et animert teksturlag tegnet av en eksisterende klasse kalt *WaterRenderPass*. Klassen holder styr på rendere og shadere, samt hva som skal reflekteres i forhold til kameraets posisjon (figur 19). I begynnelsen av implementeringen var tanken å forsøke



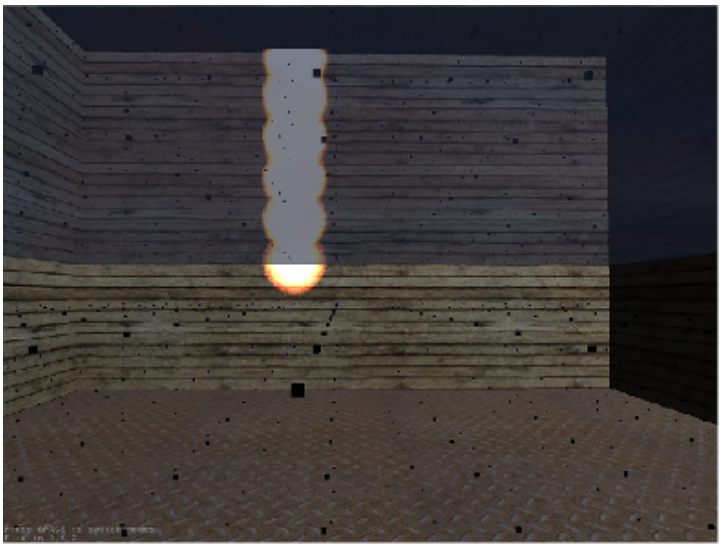
Figur 19: Vannoverflate med refleksjon

å endre fargenyansen på lyssettingen under vannoverflaten for å skape en illusjon av nedsenkning. Dette har ikke blitt implementert, da det ikke har vært relevant å operere med svært høye vannivåer. Vi anslår også at dette ville vært særdeles tidkrevende å implementere. Vannivået styres opp og ned vha. *FloodController*-klassen. I denne klassen regnes stigningshastighet ut, basert på utstrømningsvolum og rommets dimensjoner. Det er også mulig å sette et maksimalt oversvømmelsesnivå i denne klassen. I utgangspunk-

tet planla vi å ta høyde for voldsomme oversvømmelser, med spredning til naborom. Vi bestemte oss etterhvert for å legge denne idéen til side, da det stort sett er mindre oversvømmelser i form av vannlekkasjer som er relevant for oss.

4.9 Slukking av brann

Klassen *WaterfloodFire* blir benyttet til å slukke en brann hvis vannivået ved en oversvømmelse overstiger et punkt hvor det eksisterer en brann. Det sjekkes gjennom alle punktene i rommet ved hver spilloppdatering/"game loop". For hvert punkt sjekkes det om det eksisterer et tilsvarende brannpunkt. Hvis det eksisterer et tilsvarende punkt og vannivået står over punktet, sjekkes det videre om brannpunktet står i brann. Hvert brannpunkt inneholder en variabel, *inFlames*, som angir om det er en brann i punktet eller ikke. Denne settes til *false* for å slukke brannen, samtidig som partikkelutspringet i det aktuelle punktet fjernes. Da vann-nivået sjekkes mot brannens plassering på veggen legges det på en feilmargin på vann-nivået som en følge av at partiklene stikker litt på utsiden av senteret på partikkelutspringet. Se figur 20. Denne feilmarginen retter på dette



Figur 20: Feilmargin brannslukking: Viser at brannen ikke slukker før vannet når senteret av partikkelutspringet

problemet slik at det ikke fortsetter å brenne under vannoverflaten.

4.10 Meldingsdisplay

Vi lagde et enkelt tekstfelt (klassen *GameMessageField*) som kommer opp nede til venstre på skjermen. Poenget med dette er å vise kortfattet informasjon om hva som skjer, eksempelvis brukeren som klikker med musa for å starte en brann (se figur 21). Dette er bygd på et såkalt singleton pattern som sikrer at det bare er ett slik tekstfelt globalt. Men årsaken til at vi valgte å benytte singleton pattern er at dette gjør tekstfeltet mer tilgjengelig på tvers av klasser. Det er mange klasser som skal kunne bruke dette feltet og vi ville unngå at alle disse måtte instansieres med en referanse til ett og samme tekstfelt. Følgende linje kan brukes hvor som helst i koden for å vise en melding i feltet:

```
GameMessageField.getGameMessageField().displayMessage(
            "Hello, World!");
```

Figur 21: Slik ser det ut. Nyere beskjeder havner nederst.

Vi hadde tenkt at denne klassen også skulle være datastrukturen for å lagre hendelsesforløp, da alt som blir vist i meldingsfeltet forblir lagret i en datastruktur selv om de forsvinner ut av meldingsfeltet. Da måtte i såfall meldingene kobles til et tidsmerke slik at man senere kunne "spille av" et tidligere gjennomgått hendelsesforløp. Det var også tiltenkt at dette feltet skulle fungere som en enkel tekstkommunikasjon mellom personene som er tilkoblet til et scenario over nettverk/Internett. Dette vist i figur 22.

Figur 22: Slik var det tiltenkt, men det er ikke lagd funksjonalitet for dette.

4.11 Musepekerverktøy

En vanlig problemstilling når man inspiserer et scenario kan være: "Hva skjer videre hvis det oppstår en brann her?" Denne tenkte brannen oppstår kanskje grunnet en hendelse som ikke er modellert i programmet/verktøyet. Da er det ønskelig at brukeren kan starte denne hendelsen med musepekeren. Dette er løst ved å la brukeren skifte mellom "FPS-modus" og "mouseclick-modus." I FPS-modus beveger du synsretningen med musen slik man er vant til i spill. Trykker man på space vil man går over i mouseclick-modus der man får en musepeker knyttet til musebevegelsen. Når man trykker med denne musepekeren vil klikkeposisjonen bli sendt til alle registrerte "lyttere." En slik lytter tar så i mot denne posisjonen og setter i gang funksjonalitet avhengig av hva slags type lytter det er. I vår løsning inngår bare én lytter, det er en som starter branntilløp i punktet.

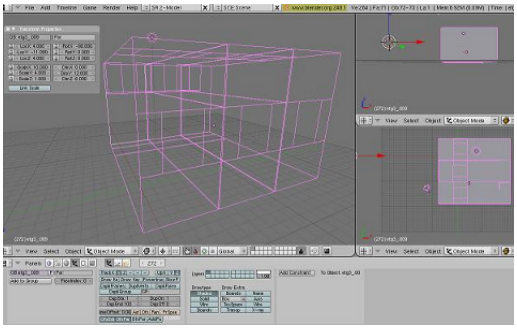
Denne funksjonaliteten har også vært veldig nyttig i forbindelse med testing siden vi har hatt bruk for å raskt plassere ut en brann i forskjellige deler av et scenario for å se

hvordan partiklene oppfører seg, eventuelt for å se hvordan brannspredningen arter seg.

4.12 Produksjon og modellering av scenarier

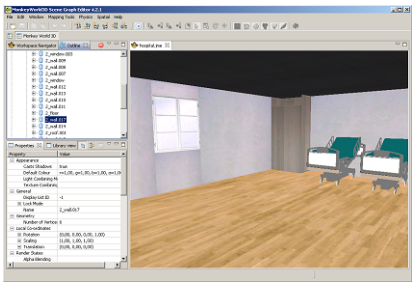
4.12.1 Utførelse

For å teste de forskjellige effektene vi trengte (ild, røyk, oversvømmelse), satte vi i startfasen av implementeringen opp et hardkodet testrom. Dette rommet var veldig enkelt sammensatt og ble brukt et godt stykke ut i utviklingsperioden. For å få presentert verktøyets hensikt og funksjon, ble det etterhvert nødvendig for oss å bygge scenarier som til en viss grad minner om omgivelser vi finner i det virkelige liv. Scenariene er blitt forenklet og nedskalert til en viss grad for å enklere demonstrere konseptene. For å konstruere disse scenariene har vi benyttet oss av 3D-modelleringsverktøyet Blender (figur 23). Ingen av gruppemedlemmene har noen tidligere erfaring med bruk av dette verktøyet. I tillegg til Blender, etterbehandles modellene med spillmotorens redigeringsverk-



Figur 23: Skjermdump av blenders modelleringsmiljø

tøy for romlige scener - MonkeyWorld 3D. Med dette verktøyet kan vi legge teksturer på alle overflater, konvertere modellformater og organisere romlige legemer i et strukturert nodetre. *Merk at teksturer absolutt bør ha en høyde og bredde som tilsvarer en toerpotens.*



Figur 24: Skjermdump av MonkeyWorld3D

For å få modeller til å fungere i verktøyet vårt er vi blitt nødt til å fastsette en spesielt utformet nodestruktur. Mer spesifikt innebærer dette at alle romlige legemer i et rom skal være innkapslet i en felles overordnet romnode. Det er også et krav at alle romlige legemer har fornuftige navn. Alle vegger skal ha strengen “wall” ett eller annet sted i navnet sitt. Det samme prinsippet gjelder for gulv og tak. Etter hvert som modeller blir ferdigstilt, blir de lagret i .jme-format, som er spillmotorens eget modellformat. Figur 25

viser en sammensmeltet illustrasjon av begge sider av et ferdig modellert scenario.

Figur 25: Sykehusscenario med serverrom, senger, ventilsjakt osv.

4.12.2 Scenarier for demonstrasjon

For å kunne presentere prosjektet vårt, og for å best vise hva det går ut på, har vi satt sammen to demo-scenarier.

4.12.3 Sykehus-scenario

Dette scenariet ble basert på et forslag vi fikk av veileder. Scenariet har to rom, fordelt på to etasjer. Dette scenariet består av et serverrom og et pasientværelse. Serverrommet er lokalisert i første etasje. Pasientværelset står rett over serverrommet, i andre etasje. Mellom disse rommene går en ventilasjonssjakt som er ment å fungere som en brannakselerator. Modellen ble satt opp med tre forskjellige infrastrukturer; vannføring, telekommunikasjon, gass. Vannføringen er reprensert i form av et rør som går gjennom serverrommet, vis-à-vis det sårbare telekommunikasjonsutstyret. Det er også plassert en gasstank som er sluttet til et rørnett i andre etasje. Denne modellen kan i praksis brukes til å demonstrere flere av elementene vi har implementert. I utgangspunktet ønsker vi å vise en hendelse der det oppstår en lekkasje i vannføringen som forårsaker en kortslutning, og dermed katastrofal svikt i telekommunikasjonsutstyret. Det er også mulig å utvide dette til at det oppstår en brann som følge av kortslutning. En slik simulasjon vil i så fall gi et innblikk i hvor katastrofale utfallene kan bli ved svikt i kritiske infrastrukturer.

Vi er blitt nødt til å låne noen rekvisittmodeller fra nettet for å få frem at hendelsene faktisk utspiller seg i et sykehusbygg. Sykesengene ble lånt fra accustudio [23], disse er publisert under en fullstendig fri lisens. Kommodemodellen ble lånt fra blogscopia [24], under “Creative Commons 3.0 Unported”-lisens [25].

4.12.4 Kabelbrann-scenario

I november i 2007 ble Oslo Sentralstasjon rammet av brann etter at en entrepenør hadde gravd over en høyspentkabel. Overgravingen førte til at en kabel kortsluttet ved en kabelskjøt i en kabeltrasé. Brannen førte til full evakuering og strømstans. All togtrafikk ble stanset, og driften ble ikke gjenopprettet før sent neste dag. Omlag 80 000 reisende ble berørt. I kabeltraséen gikk det også fiberføringer. Omlag 25 000 internett-kunder, noen telefonkunder og kunder av leide samband var uten tjenester i omlag 10 timer som en følge av skader på disse [17]. Samlokalisering av samfunnskritiske kabler er en problemstilling vi mener er svært aktuell og relevant for oss. Vi har derfor valgt basere vårt andre demo-scenario omkring dette.

5 Testing og kvalitetssikring

5.1 Kvalitetssikring

I prosjekter av denne størrelses- og viktighetsgrad vil det være fatalt å miste noe produsert materiale. Tap av data vil føre til et langt skritt i feil retning, og kan få store konsekvenser for gjennomføringen av prosjektet. Vi fikk tildelt lagringsområde på skolens server, og satte opp SVN-klient med versjonshåndtering opp mot dette området både for dokumentasjonen og kildekoden. På denne måten vil alle til enhver tid ha tilgang til siste versjon av dokumentasjon og kildekode. Lagringsområdet ligger under skolens rutiner for backup, så vi ville uansett ikke stått på bar bakke hvis det skulle hende noe med serveren. Vi utnevnte allikevel i begynnelsen av prosjektet en backup-ansvarlig med ansvar for å ta backup av lagringsområdet én gang i uken, samt brenne denne på en CD. På denne måten ville vi ha minst én backup hvis det skulle dukke opp en kritisk hendelse som førte til tap av data.

5.2 Testing

Vår opprinnelige testplan innebefattet testing etter hver fullførte sprint i prosjektet. Underveis i implementeringen erfarte vi at en fulltest av systemet annenhver uke ikke ville være mulig å gjennomføre. Dette som en følge av at fremdriftsplanen og prioriteringer ble endret underveis i prosjektet. Vi valgte derfor å benytte oss av empirisk testing, som ofte omtales som “prøve- og feilemetoden”. Dette vil si at vi har testet koden vår ved å kompilere den, for så å visuelt inspisere hvordan det oppfører seg. Dette gir et raskt svar på hvordan forskjellige tilnærminger og operasjoner spiller inn på programmets virkemåte.

Vi har konsentrert oss om at de forskjellige delene av implementasjonen har fungert i testrommet vi satte opp til å begynne med. De forskjellige delene av implementasjonen har blitt kontinuerlig testet både som et ledd i å forstå hva som skjer, og selvfølgelig for å sjekke at det fungerer som det skal.

Ulempen ved denne typen testing er at vi kun vet hvordan tingene oppfører seg i testrommet, og ikke noe om hvordan det vil fungere i rom som har en annen karakteristikk/oppbygning.

Etter hvert som flere deler falt på plass og fungerte hver for seg, ble testene mer sammensatte og komplekse. Det var i løpet av disse testene vi begynte å få en forståelse for hvordan hele systemet fungerte sammen, og om våre valgte løsningsmetoder var gode nok. Videre kunne vi nå også teste de forskjellige elementenes innvirkning på hverandre, og om samspillet mellom disse var naturlig og realistisk.

Etter hvert som prosjektet skred frem så vi nytten av å ha noen hjelpeverktøy til for å inspisere modellen/omgivelsene og hva som faktisk skjedde *mens* programmet kjørte. Klassen *TemperaturePoints* er en slik hjelpeklasse som inneholder funksjonalitet for å legge til et synlig punkt i rommet for hvert punkt i temperaturfeltet. Disse punktene forandrer farge basert på temperaturen og hjalp på den måten å se hvordan temperaturen i rommet endret seg over tid.

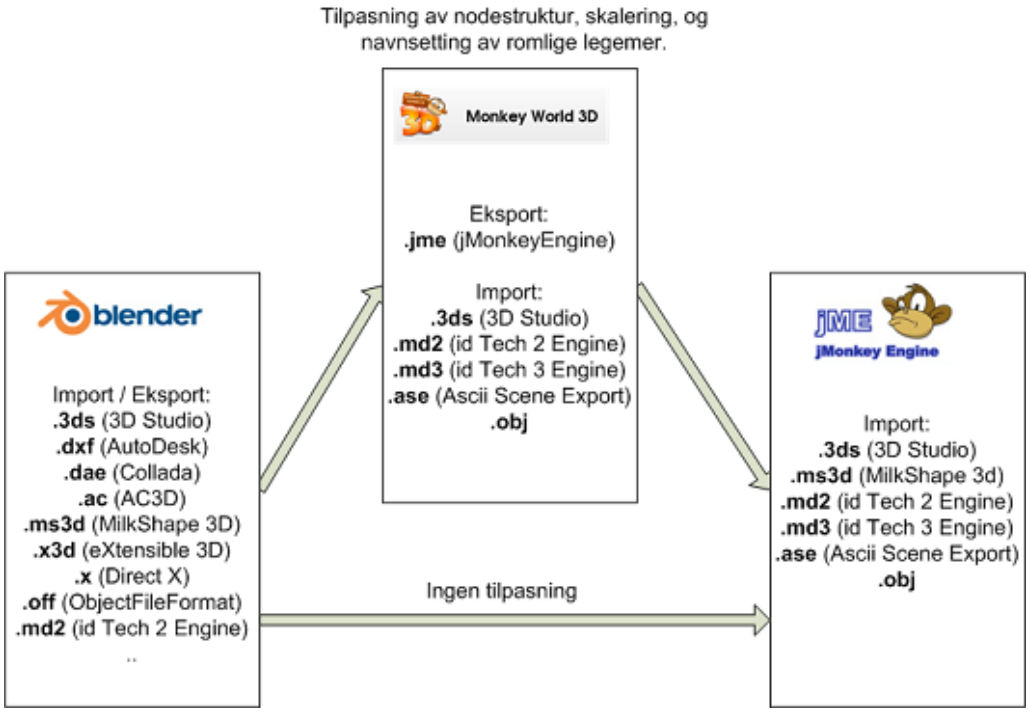
Klassen *VectorArrow* er et annet hjelpemiddel vi lagde for å visualisere vektorfeltet og hvordan det endrer seg under kjøring. Dette er rett og slett en pyramide/pil som peker i samme retning som tilhørende vektorstørrelse.

MouseViewerPicker er nevnt tidligere. Denne ble laget delvis som en del av brukergrensesnittet, men det var også et nyttig hjelpemiddel for oss spesielt i tilknytning til branneffekten. Denne funksjonaliteten hjalp oss med å plassere ut brann på en valgfri posisjon i scenen og hjalp oss på denne måten å raskt se hvordan samspillet mellom ting fungerte.

6 Installasjon og realisering

6.1 Innlasting av modeller

I utgangspunktet var det ønskelig at man skulle kunne importere modeller lagret i et CAD-format. Det er mulig å laste inn slike modeller, og sette opp scenarier i disse. Dette krever likevel noe konvertering og tilpasning siden jMonkeyEngine ikke støtter CAD-formater direkte. Figur 26 viser hvordan dette henger sammen.



Figur 26: Import-/eksportoversikt

Et veldig vanlig CAD-format er .dxf-formatet. Dette er et format som støtter 3D-tegning. Formatet kan importeres til Blender via et Python-script som følger med i installasjonen. Derfra er det mulig å eksportere til et format støttet av jMonkeyEngine og MonkeyWorld 3D. For å tilpasse modellene, og for å sørge for at nodetreet og navnene i modellen er lovlige for bruk med vårt verktøy, kan modeller importeres inn i MonkeyWorld 3D. MonkeyWorld 3D er et verktøy som muliggjør redigering av 3D-modeller i sanntid med bruk av jMonkeyEngine. MonkeyWorld 3D lagrer ferdigredigerte filer i .jme format, som er jMonkeyEngines eget filformat.

6.2 Installasjon

På cd-en ligger en mappe kalt jar. Denne inneholder det som trengs for å kjøre programmet, en egen installasjonsprosess er ikke nødvendig. Man kopierer mappen komplett med undermapper over til sin pc og eksekverer.

6.3 Eksekvering

For å kjøre programmet utenfor Eclipse har vi laget en jar-fil. Dette er et arkiv som inneholder alt av teksturer, modeller og binære klassefiler. For å lage en slik jar-fil har vi tatt i bruk en tilleggsmodul for Eclipse som heter Fat Jar Eclipse Plug-In [14]. Vi fulgte fremgangsmåten som beskrevet på nettsiden for verktøyet og fikk ut én frittstående fil. Når denne startes må den få vite hvor LWJGL ligger. Dette har vi lagt på cd-en i samme mappe som jar-filen. Programmet startes med følgende kommando: *"java -Djava.library.path=lwjgl/native/windows -jar vciige.jar"*. Bytt ut "windows" med "linux" avhengig av hva slags operativsystem som kjøres. Dette er testet under Windows XP og Ubuntu 9.04 og virker. Under begge operativsystemer var det installert Java Development Kit og Java-versjonen var 1.6.0-13.

7 Beskrivelse av utviklingsprosessen

7.1 Vår anvendelse av Scrum

Vi har valgt å benytte oss av noen grunnleggende elementer i Scrum som utviklingsrammeverk. Det ble tidlig bestemt at modellen ikke skulle følges slavisk. Følgende elementer har vært aktuelle for oss:

- Sprint-inkrementer (to til fire uker).
- Daglig prosjektstatusmøte internt i gruppen.
- Samlokalisering og muntlig kommunikasjon.
- Sprint-møter (planlegging og vurdering) før hver sprint.
- Levering av synlig/fysisk produkt etter hver sprint.

Vi har hele tiden jobbet med to-ukers sprint-inkrementer. Disse er ikke brutt ned i mindre inkrementer, slik Scrum-modellen i utgangspunktet foreslår. Veilederen vår har ikke alltid vært tilgjengelig for fysiske møter, og selv om han har vært tilgjengelig via IP-telefon har vi valgt å skyve litt på disse møtene for å enklere kunne gå igjennom de forskjellige problemstillingene sammen. Når vi fikk organisert et møte med veileder gikk vi igjennom planer for neste sprint og eventuelt det som gjensto. Vi gikk samtidig gjennom resultatet av forrige sprint. Etter hver sprint sendte vi ut demonstrasjonsvideoer til veileder og oppdragsgiver. Dette slik at de kunne se progresjonen i prosjektet med egne øyne, og gi konstruktive tilbakemeldinger. Disse videoene ble også publisert på prosjektets hjemmeside. Vi har ikke benyttet oss av noen dokumentartefakter i Scrum-modellen. Dette er fordi vi føler gannt-skjemaet, arbeidsloggen og muntlige enigheter i gruppen har vært tilstrekkelig.

7.2 Utvikling av systeminkrementer

I startfasen av hvert systeminkrement, tok vi i betraktning de justeringer og forbedringer som ble avtalt på det foregående sprint-møtet. Det kunne forekomme at sprintene ble noe avkortet eller forskjøvet/byttet etter resultatet av det foregående sprint-møtet. Når implementeringen av et inkrement skulle gjennomføres, gikk vi alltid igjennom kildekoden i jMonkeyEngine for å finne eksempler som kunne være til nytte for oss. Dette har helt klart vært en tidsbesparende faktor for oss i noen tilfeller. Vi stilte også spørsmål på forumet på jMonkeyEngines hjemmesider dersom vi sto fast i vanskelige spillmotortekniske problemstillinger. Relevant kommunikasjon fra forumet ligger lagret på den medfølgende cd-platen.

7.3 Arbeidsfordeling

Foran hver sprint ble vi i gruppen enige om oppgavefordelingen. Stort sett løste dette seg ved at alle gruppemedlemmene plukket ut en oppgave som var interessant og som føltes realiserbar av den enkelte. Dette er en filosofi som Scrum-modellen støtter, da dette bidrar til større produktivitet og sikkerhet.

8 Diskusjon av resultater

8.1 Resultater

Det var i utgangspunktet vanskelig for oss å sette en klar målsetting for akkurat hva vi skulle oppnå og hvor langt vi skulle komme. Ingen av oss hadde noen som helst erfaring med spillmotor-programmering i forkant av prosjektet, og vi hadde slikt sett ikke noe grunnlag for å gjøre en nøyaktig estimering. Det har vært utfordrende å gå fra å skrive desktop-applikasjoner til å programmere i tre dimensjoner. Spillmotor-biblioteket er omfattende og inneholder hundrevis av klasser og metoder som bygger på konsepter og emner vi ikke har studert tidligere. Vi var ambisiøse under utarbeidelsen av forprosjektet og satte oss en målsetting om å produsere en programvare som skulle være komplett, og klar til bruk.

Vi har etterhvert som prosjektet har utviklet seg måttet moderere målsettingen vår. Etter en måneds tid ut i utviklingsfasen ble vi enige med oppdragsgiver om at prosjektet skulle bli et "proof of concept". Vi bestemte oss for at vi ville vise at det lar seg gjøre å utvikle dette verktøyet vha. fri programvare. Problemløsning og bugfiksing underveis har vært tidkrevende. I utgangspunktet ønsket vi et intuitivt brukergrensesnitt, opptaksmuligheter og flerspillermuligheter over nettverk. Det har hele tiden vært fokus på å få til en brukbar tilnærming til virkeligheten i simuleringene, og de andre modulene har derfor måttet vike til fordel for det. Dette har blitt bestemt under møter med veileder og oppdragsgiver underveis. Det har også underveis blitt brukt ressurser på utvikling av moduler som senere har blitt forkastet. Det ble blant annet brukt en del tid på å utvikle funksjonalitet for å sørge for at vinduer kollapser ved høyt trykk. Dette ble mer eller mindre forkastet, da det ikke var aktuelt å introdusere slike vannmengder under simulering. Her måtte jMEPhysics introduseres i stor grad. Detaljer som dette har ofte vært svært tidkrevende å implementere, da det har vært en del nye konsepter og verktøy som har måttet bli tatt i bruk for å få realisert funksjonaliteten. Brannmodulen er et godt eksempel på en tidkrevende prosess. Her måtte vi først finne en måte å introdusere temperatur i et gitt rom på. I virkeligheten forandrer temperaturene i de ulike delene av et rom seg, hele tiden. Så lenge vi ikke har et vakuum i dette rommet, har vi vind/trekk som bidrar til temperaturforflytningen i rommet. Trekken i rommet, og temperaturen, er også i stadig endring. Det å få konvertert disse grunnleggende elementene fra virkeligheten til fungerende kildekode, har vært svært tidkrevende og utfordrende.

Vi har allikevel kommet i land med det vi definerer som verktøyets kjernefunksjonalitet; selve visualiseringen av forstyrrelser i kritiske infrastrukturer som resulterer i brann, røykspredning, og lekkasjer/oversvømmelser. Ut over det er det også laget funksjonalitet for innlasting av bygningsmodeller.

8.2 Muligheter og valg underveis

8.2.1 Valg av utviklingsspråk

Vi har benyttet oss av Java som utviklingsspråk i dette prosjektet. Ut fra de forutsetningene vi hadde i forkant av prosjektet sto valget mellom C++ eller Java. Det er stort sett

C++ som er brukt i spillsammenheng, men vi fant også et alternativ basert på Java. Grunnen til at vi endte opp med å velge Java var at vi nettopp hadde fullført et kurs i programutvikling, og vi hadde derfor dette friskt i minnet og regnet med at vi ville tjene på dette tidsmessig. Vi vurderte også Java for å være et godt alternativ, muligens det beste, med tanke på plattformfleksibilitet.

8.2.2 Valg av spillmotor

Valget falt her på jMonkeyEngine som var det eneste veletablerte, Java-baserte alternativet vi kunne finne. Det stod for oss mellom denne, CrystalSpace 3D og Ogre (begge to basert på C++). Vi fant veldig mange eksempler på prosjekter basert på jMonkeyEngine, alt fra enkle studentprosjekter til semiprofesjonelle spill. Til og med Intelligent Robotics Group, NASA har benyttet seg av spillmotoren (presentert på JavaOne 2008). Vi så at det var mye aktivitet på spillmotorens hjemmesider hvor alle er velkomne til å stille spørsmål og få hjelp. I tillegg leste vi en del brukererfaringer og omtaler rundt motoren som nesten utelukkende var positive, og mange nevnte at brukermiljøet rundt var vennlig og svært hjelpsomt, også for nybegynnere.

Kravene var at det måtte være nettverksstøtte, dette er tilgjengelig enten gjennom Project Darkstar eller JGN. Det måtte være tilrettelagt for effekter som vann, ild, osv., dette var på plass - i tillegg fant vi en egen editor for slike partikkeleffekter. Så vi endte opp med å velge jMonkeyEngine sammen med de tilhørende bibliotekene jMEPhysics og JGN.

8.2.3 Valg av utviklingsverktøy

Eclipse Java IDE har innebygget støtte for SVN ved installasjon av en ekstern plugin. Dette gjør at vi kan holde orden på nyeste versjoner, og versjonshistorie, av vårt prosjekt. Samtidig ga dette oss muligheten til å hente inn jMonkeyEngine, JGN og jMEPhysics direkte inn i Eclipse. Et alternativ hadde vært å benytte oss av en vanlig tekstbehandler og en ekstern SVN-klient, men her hadde vi mistet oversikten over de forskjellige delene av prosjektet. I Eclipse får man derimot meget god oversikt, og JavaDoc for klasser og funksjoner kommer opp automatisk. Dette gjør at man slipper å bruke unødvendig mye tid på å lete etter dokumentasjonen. Vi hadde opprinnelig planlagt å bruke Apache Ant og Doxygen pluss Eclox Hot i prosjektet vårt, men etter hvert gikk vi bort fra disse. Vi så ikke behovet for å generere dokumentasjon med Doxygen når vi kunne bruke JavaDoc. Vi hadde god kontroll over kodedelen av prosjektet ved å bruke Eclipse, Eclipses SVN-plugin og Fat Jar.

8.2.4 Valg av tekstbehandlingsverktøy

Vi har stort sett benyttet oss av LaTeX til dokumentasjon av prosjektet. Vi hadde liten kjennskap til LaTeX fra før, men vi valgte å benytte oss av dette da det forenkler versjonshåndteringen. Dokumenter får etter vår mening et flott utseende og en profesjonell layout samtidig som man nesten utelukkende kan konsentrere seg om innholdet.

8.3 Forslag til videre utvikling

Siden det meste av kjernefunksjonaliteten er på plass vil det være store muligheter for videreutvikling ved en senere anledning. Det gjenstår en del elementer for at dette skal kunne være et komplett og ferdigstilt verktøy.

Det kan med fordel implementeres en modul som muliggjør endring av materiale-

genskaper på omgivelser for å forenkle prosessen med å sette opp et scenario. Vi ser for oss en enkel utvidelse av museklikk-klassen vår for å få til dette. Slik det er nå må man kjenne til koden for å tilegne materialegenskaper til objekter. For å reelt sett kunne bruke verktøyet trengs et brukergrensesnitt, og en eventuell nettverksmodul for samkjøring via nettverk. Disse modulene er funksjonalitet som ikke burde være spesielt tidkrevende å implementere slik vi ser det, men dette er blitt bortprioritert til fordel for kjernefunksjonaliteten i verktøyet.

Navigeringsmuligheter i form av menyer ved oppstart og eventuelt høyreklikking inne i "spillet" vil gjøre at verktøyet blir mer helhetlig. Herunder vil det også være nødvendig å implementere den funksjonaliteten som hører til hvert valg på de forskjellige menyene.

Vi har tidligere vært inne på at vi har bortprioritert en del ting til fordel for kjernefunksjonaliteten, og disse elementene vil være høyst aktuelle ved en eventuell videreutvikling. Innlesing av bygningsmodeller direkte fra fil med tilhørende konverteringer forbundet med dette er et viktig element for at verktøyet skal kunne ha en direkte nytteverdi for bedrifter. Opptaksfunksjon for forskjellige scenariegjennomganger vil være høyst relevant for å kunne bygge opp under en risikoanalyse.

Legge til støtte for strukturert kabling. Det vil si å legge til funksjonalitet som importerer informasjon om hvordan kablingen er i et bygg: posisjon, lengde og oppgave. Denne informasjonen må inn i scenariet slik at effektene kan påvirke kabelavhengigheter. Dette er en viktig faktor i kritiske infrastrukturer med tanke på kabelsmelting og andre kabelbrudd.

9 Evaluering av gruppens arbeid

9.1 Innledning

Da tiden var inne for å danne grupper til bachelorprosjektet var valget for oss enkelt. Vi har i løpet av utdanningsløpet vårt ved HiG jobbet sammen ved en rekke anledninger, og kjenner hverandre godt i prosjektsammenheng. Erfaringene vi har gjort oss gjennom tidligere samarbeid har ført til at fordelingen av arbeidsoppgavene i dette prosjektet ble noe lettere. I et så betydningsfullt prosjekt som bachelorprosjektet har det vært en ekstra trygghet å jobbe sammen med personer man i utgangspunktet vet fungerer bra sammen, samt har potensiale til å gjøre et godt stykke arbeid.

9.2 Organisering

Vår gruppe har jobbet sammen ved forskjellige prosjekter gjennom studiet og kjenner hverandres kapasiteter og potensial. Vi valgte gruppeleder i startfasen av prosjektet som skulle ha muligheten til å skjære gjennom hvis uenigheter i gruppen ikke lot seg løse. Det har i løpet av prosjektperioden blitt noen faglige diskusjoner, men disse har resultert i enighet innad i gruppen uten nevneverdige problemer.

9.3 Fordeling av arbeidet

Systemets inndeling har gjort fordelingen av arbeidet til en grei oppgave. Vi har i forkant av hver sprint gått gjennom i plenum hva som skal gjøres, og diskutert oss frem til én eller flere mulige løsningsmåter.

Det har naturlig nok vært deler av systemet som har vært betydelig mer vanskelig å implementere enn andre. Problemer som har oppstått i forbindelse med utviklingen har i hovedsak blitt løst felles; enten i form av parprogrammering eller diskusjon og tavlebruk.

9.4 Prosjekt som arbeidsform

Vi synes arbeid med prosjekt gjør det lettere å tilegne seg ny kunnskap. Selv om hvert enkelt gruppemedlem har fulgt samme utdanningsløp, vil kompetansen på de forskjellige delene avvike noe ut fra interesser etc. Dette gjør at man som gruppe kan dra nytten av hverandres kompetanse, og dermed øke sin egen kompetanse rundt temaet.

Prosjektarbeid leder gjerne til en del diskusjoner når det kommer til valg av løsningsmetoder etc. Disse diskusjonene gjør at hver enkelt må reflektere over de forskjellige forslagene som blir skissert, samt vurdere positive og negative sider ved disse.

9.5 Tilbakeblikk på risikoanalysen

Som et ledd i forprosjektet utførte vi en risikoanalyse der vi prøvde å identifisere mulige hendelser/situasjoner som kunne ha en negativ innvirkning på prosjektet vårt. Her vil vi diskutere hva som skjedde og hva det hadde å si.

Langvarig sykdom: Dette har ikke inntruffet. Ingen av gruppemedlemmene har hatt annet langvarig fravær heller.

Svikt i maskinvare: Hans Jacob fikk problemer med et defekt skjermkort på sin stasjonære pc, noe som førte til en del ekstraarbeid og irritasjon. Dette ble ikke kritisk, siden han har en rimelig kraftig bærbar pc i bakhånd, men uansett gikk det bort noe tid på dette.

Ikke bli ferdig med funksjonaliteten som skal leveres på sprinten: Etter hvert som prosjektet gikk fremover, innså vi at den opprinnelige tidsplanen var vel ambisiøs. Det er mye av funksjonaliteten som vi ønsket å lage som vi ikke fant tid til. Dette har ført til at vi har justert tidsplanen underveis, og vi har snakket om hvor fokus burde rettes videre på veiledningsmøtene vi har hatt.

Manglende kompetanse: Systemutvikling sammen med spillmotorer krever en god romoppfatning og gode ferdigheter med vektorregning. Matematikkfagene vi har hatt var, etter vår mening, tilstrekkelig som forkunnskaper i så måte. Med vår bakgrunn hadde vi også god nok kompetanse rent programmeringsteknisk. Vi hadde ikke god nok kompetanse på generell bruk av spillmotorer, derfor måtte vi lære oss alt underveis. Vi har også følt at vi kunne trenge fysikkekspertise i dette prosjektet når det gjelder forenkling og modellering av f.eks. brann.

"Buggy" kode: Det har som forventet oppstått mye bugs i koden underveis, dette har blitt løst fortløpende og vi har ikke hatt noen nevneverdige problemer i tilknytning til dette.

Overskride prosjektets tidsfrist: På en måte kan man si at dette allerede har skjedd. Vi har ikke kommet i mål med alt vi ønsket å utvikle. Men alt ser ut til at vi skal komme i mål tidsnok med det vi sammen med veileder har blitt enige om å fokusere på.

Ende opp uten et demonstrerbart produkt: Vi ble tidlig enige om at videoer er en god måte å vise fram arbeidet vårt på. Spesielt med tanke på presentasjonen så er det viktig å ha video å vise fram fremfor en live-presentasjon. Vi har underveis produsert en rekke videoer, så denne fallgruven har vi styrt unna.

Vanskelighetsgrad er høyere enn antatt: Vi gikk til denne oppgaven fordi den virket spennende og utfordrende og vi var forberedte på en moderat til høy vanskelighetsgrad. Men etter hvert som vi kom mer inn i oppgaven gikk det opp for oss hvor vanskelig det er å modellere og visualisere bl.a. brann på en realistisk måte. Oppgaven var en god del vanskeligere enn vi hadde sett for oss.

9.6 Subjektiv opplevelse av bachelorprosjektet

Vår erfaring fra prosjektarbeid har vært fra mindre prosjekter i forskjellige emner her på HiG, og ingen av disse har vært i nærheten av bachelorprosjektet i størrelse og kompleksitet. Prosjekt som arbeidsform var ikke ukjent, men vi har i løpet av dette prosjektet tilegnet oss nye kunnskaper og erfaringer innen blant annet prosjektstyring.

Vi mener at vi i løpet av bachelorprosjektet har lært hva det virkelig innebærer å jobbe sammen i en gruppe. En av de store utfordringene er at man jobber for en oppdragsgiver som sitter med en idé, og vår oppgave er å oppfatte denne idéen riktig og ta dette videre til et sluttprodukt. Vi har her fått erfaring med alt fra planlegging, kravspesifisering, utvikling og dokumentering. Dette gir et innblikk i hvordan hverdagen vil se ut for oss

da vi kommer ut i arbeidslivet, da prosjekt er en utbredt arbeidsform innen databransjen. En annen ting vi har fått kjenne litt på er gruppedynamikk, hvordan personer på gruppa kan påvirke hverandre og hvordan frustrasjon/begeistring kan "smitte over" til andre gruppemedlemmer og påvirke produktiviteten.

I utgangspunktet hadde vi ikke sett for oss en så kompleks og sammensatt oppgave som dette har vært da vi leste oppgaveteksten. Mye av årsaken til dette var vår manglende forkunnskap om emnet, og derav dårlig grunnlag for å kunne estimere hvor lang tid de forskjellige delene ville ta å utvikle.

Programmeringsmessig hadde vi et godt utgangspunkt allerede før prosjektet, men vi har underveis i prosjektet plukket opp en god del tips og triks gjennom å lese kildekoden i spillmotoren. Systemet er den største implementasjonen vi har jobbet med i løpet av studiet, og vi har lært mye om kodestrukturering og hvordan lage god dokumentasjon av koden. Det ble sagt tidlig at det var ønskelig at vårt arbeid skulle danne grunnlag for videre prosjektarbeid, så vi fokuserte hele tiden på å lage leselig, oversiktlig kildekode som det er mulig for andre å forstå.

Vi har hatt jevnlig kontakt med oppdragsgiver og veileder underveis i prosjektperioden. Det har vært en god del korrespondanse på e-post pga. at vår veileder kun er tilstede på HiG noen få dager med jevne mellomrom. Vi har fått god veiledning under hele perioden, og det har ikke vært mangel på tips og entusiasme fra både veileder og oppdragsgiver. Veileder og oppdragsgiver har også sørget for å gi tilbakemelding på hvilke områder det er ønskelig å fokusere på. Vi mener at vi har hatt et godt fungerende samarbeid med både veileder og oppdragsgiver gjennom hele perioden.

10 Konklusjon

Etter vi valgte denne oppgaven hadde vi en visjon om å lage et ferdig produkt med mye av funksjonaliteten som finnes i et spill med flerspillerdel. Dette i tillegg til kjernefunksjonaliteten som er å visualisere kritiske hendelser og deres innvirkning på infrastrukturer. Nå er vi ferdige og har underveis måttet legge til side mye av funksjonaliteten vi ønsket å lage. Det har vært frustrerende å innse at det er deler man ikke klarer å levere. Når vi ser tilbake på prosjektperioden synes vi allikevel at vi har vært innom mange temaer og problemstillinger og at vi har fått til mye. Vi har lært mye om systemutvikling, det å strukturere kildekode og å dele opp programmer i fornuftige deler. Vi har også fått erfaring med å jobbe for en oppdragsgiver og utarbeide krav i samarbeid med denne. Alt dette er nyttige ting vi tar med oss ut i arbeidslivet. Vi synes også vi har fått god grunnforståelse for hvordan man programmerer spill.

Å lage effekter som skal være nære virkeligheten synes vi har vært vanskelig og tidkrevende. Det er en fin erfaring å ta med seg. Nå vet vi at høy realisme i en modell krever voldsomt mye arbeid, mer enn vi trodde, og veldig gode kunnskaper innenfor en rekke felt. Dette var jo i utgangspunktet en oppgave i kategorien informasjonssikkherhet, men vi mener at utviklergruppen som en helhet må ha veldig gode kunnskaper i både fysikk og spillprogrammering i tillegg for å løse oppgaven med tilfredsstillende realisme.

Vi sitter igjen med inntrykk av at oppdragsgiver og veileder er fornøyd med det vi har fått til og det vi har presentert. Selv synes vi at vi har fått gjort en god del og vi håper at rapporten vår kan være med å legge grunnlaget for videre arbeid på dette verktøyet og dette temaet. Videre arbeid ser vi for oss vil være å optimalisere og forfine de metodene og konseptene vi har utviklet slik at den totale ytelsen blir bedre. Videre bør det legges til en flerspillerdel og funksjonalitet rundt dette. Ved å bruke open source-teknologier har vi gjort det så åpent og tilgjengelig som mulig for andre å bygge videre på.

Dette skulle være et proof of concept og vi mener at utviklingen av et slik verktøy kan gjennomføres. Når tre personer som er totalt nybegynnere innenfor spillmotorer kan få til det vi har fått til mener vi at dette er absolutt gjennomførbart hvis det blir satt flere utviklere på prosjektet og det ikke er knapphet på tid.

Bibliografi

[1] Ian Sommerville, *Software Engineering*. Addison Wesley, 8th Edition, 2007.

[2] Randall D. Knight, *Physics for scientists and engineers*. Addison Wesley, 2004.

[3] Harvey and Paul Deitel, *Java how to program*. Prentice Hall, 7th Edition, 2007.

[4] Sun Microsystems,
Code Conventions for the Java Programming Language.
`http://java.sun.com/docs/codeconv/`
(sist besøkt 20.05.09)

[5] jMonkeyEngine,
BSD license.
`http://www.jmonkeyengine.com/wiki/doku.php?id=bsd_license`
(sist besøkt 20.01.09)

[6] LaTeX Editor (LEd),
`http://www.latexeditor.org/`
(sist besøkt 20.04.09)

[7] Blender,
`http://www.blender.org/`
(sist besøkt 20.05.09)

[8] jMonkeyEngine,
`http://www.jmonkeyengine.com/`
`http://jmonkeyengine.googlecode.com/svn/trunk/`
(sist besøkt 20.05.09)

[9] Java Game Networking (JGN),
`http://jgn.googlecode.com/svn/trunk/`

[10] jME Physics,
`http://jmephysics.googlecode.com/svn/trunk/`

[11] The Eclipse Foundation,
Eclipse.
`http://www.eclipse.org/`
(sist besøkt 30.03.09)

[12] Apache Ant,
`http://ant.apache.org/`
(sist besøkt 16.02.09)

[13] Doxygen,
`http://doxygen.org/`
(sist besøkt 16.02.09)

[14] Fat Jar Eclipse Plug-In
http://fjep.sourceforge.net
(sist besøkt 13.05.09)

[15] Monkey World 3D
http://mw3d.org/
(sist besøkt 20.05.09)

[16] Gassinnhold,
http://www.kj.uib.no/sikkerheitshandbok/word_html/kap6
(sist besøkt 05.03.09)

[17] Rapport: Brann i kabelkulvert –Oslo Sentralstasjon 27.11.2007,
Direktoratet for samfunnssikkerhet og beredskap
(sist besøkt 04.05.09)

[18] Steven M. Rinaldi, James P. Peerenboom, and Terrence K. Kelly, *Identifying, Understanding, and Analyzing Critical Infrastructure Interdependencies*. IEEE Control Systems Magazine, December 2001.

[19] NOU. *Når sikkerheten er viktigst* 2006

[20] Fire Dynamics Simulator (Version 5): User's Guide,
McGrattan, K. B.; Klein, B.; Hostikka, S.; Floyd, J. E.
NIST
http://www.fire.nist.gov/bfrlpubs/fire07/PDF/f07053.pdf
(sist besøkt 18.05.09)

[21] St. Julien, Tazama U. and Shaw, Chris D.
Firefighter command training virtual environment
TAPIA '03: Proceedings of the 2003 conference on Diversity in computing

[22] Mittring, Martin and GmbH, Crytek
Advanced virtual texture topics
SIGGRAPH '08: ACM SIGGRAPH 2008 classes

[23] Accustudio
http://www.accustudio.com/index.php?option=com_remository&Itemid=0&func=startdown&id=125
(sist besøkt 19.05.09)

[24] Blogscopia
http://resources.blogscopia.com/models/mod_interior_en.html
(sist besøkt 19.05.09)

[25] Creative Commons 3.0 Unported License
http://creativecommons.org/licenses/by/3.0/
(sist besøkt 19.05.09)