Hovedprosjekt:

DIRUS

Disasters 'R' Us

Virtuelt spillbord for Sivilforsvaret

FORFATTERE:

Per Kristian Juvstad
Andreas Holen
Hans Olav Skogstad
Haakon Aarsby

Dato: *19. mai 2004*

Sammendrag av hovedprosjekt

Tittel:	Dirus (Disasters 'R' Us)	Nr. : 13 Dato : 19.05.04
Deltakere:	Per Kristian Juvstad Hans Olav Skogstad Andreas Holen Haakon Aarsby	
Veileder:	Tom Røise	
Oppdragsgiver:	Sivilforsvaret Starum	
Kontaktperson:	Mads Ringen	
Stikkord (4 stk)	Virtuelt ulykkesscenario, 3D-grafikk, 3D-modellering, C/C++	

Antall sider:	Antall bilag:	Tilgjengelighet (åpen/konfidensiell):

Kort beskrivelse av hovedprosjektet:

Vi har laget et virtuelt ulykkesscenario for Sivilforsvaret. Sivilforsvaret bruker i dag fysiske modellbord for å trene opp beredskapspersonell, men håper på å gå over til å bruke virtuelle scenarier i denne opplæringen. Dette virtuelle ulykkesscenariet som vi har utviklet skal brukes for å demonstrere noen av mulighetene man har til å utvikle et virtuelt landskap og virtuelle hendelser med dagens teknologi, og skal kunne brukes som grunnlag for å utvikle en full virtuell opplæringsapplikasjon. Vi har utviklet en prototype som er basert på et førstepersons-skytespill. Denne prototypen er utviklet i C/C++ og benytter en 3D grafikkmotor, Genesis3D, for å generere grafikk. Scenariet foregår i et 3D-landskap og inneholder dynamiske elementer, som f.eks. branner, røyk, mennesker og biler. Det er to forskjellige roller man kan har i scenariet. Disse to er skadestedsleder og kontrollør. Skadestedsleder er den som skal opplæres og kontrolløren er læreren. Kontrolløren starter en server og skadestedsleder logger seg på denne serveren via et lokalnettverk eller Internett. Brukerne av applikasjonen kommuniserer med mikrofon og høyttalere.
Under utviklingen har vi lagt mest vekt på funksjonalitet og ytelse. Detaljer i grafikken er ikke så viktig for sivilforsvaret siden prototypen kun er en demonstrasjon.
Web siden for porosjektet: http://hovedprosjekter.hig.no/v2004/data/gruppe13/ .

Forord

Det virtuelle spillbordet Dirus er utviklet av fire tredjeårsstudenter ved datalinjen på Høgskolen i Gjøvik. Vi valgte denne oppgaven på grunn av dens utfordrende og spennende natur på bakgrunn av oppdragsgivers oppgavebeskrivelse. Vi synes også det er spennende å jobbe med 3D-teknologi, og å få vist teknologiens muligheter. Oppgaven vil gi gruppens medlemmer en viktig erfaring innen programmering og prosjektstyring.

Vi ønsker å rette en stor takk til kontaktperson Mads E. Ringen, for sitt engasjement og gode humør. Vi har alltid hatt en god dialog over e-post, og de vi har møttes på HiG og Starum. Vi retter også en takk til Bernt Haakon Danielsson for å ha vært vår kontaktperson ved spørsmål angående modellbordet.

En stor takk rettes også til Tom Røise for å ha vært en god veileder under prosjektet. Når vi har stått fast tidlig i prosjektet har han alltid vist oss retningen det var viktig å konsentrere utviklingsarbeidet i. Tom er også en av skolens mest erfarne innen systemutvikling, og delt mye av sin kunnskap med oss. Under skriving av denne rapporten har han vært til uvurderlig hjelp. Vi har satt stor pris på all ris og ros.

I tillegg vil vi rette en generell takk til alle på Genesis3D-forumet, for å ha tatt seg tid til å svare på alle spørsmål vi har postet der i løpet av våren, og en generell takk til alle vi har glemt!

Gjøvik, 19. mai 2004

Per Kristian Juvstad
Prosjektleder

Haakon Aarsby

Andreas Holen

Hans Olav Skogstad

Innholdsfortegnelse

Figuroversikt

1 Innledning

1.1 Problemområde

Sivilforsvaret er representert i Rogaland, Sør-Trøndelag og Oppland (på Starum). Avdelingen på Starum (Sivilforsvarets skole Starum (SFSKS)) driver blant annet med opplæring av beredskapspersonell (politi-, brannvesen-, ambulansepersonell). De bruker fysiske modellbord, miniatyrbiler, menneskefigurer og små kamera for å simulere ulykker og trene på beredskapspersonellets oppgaver.

SFSKS ønsker å kjøre ulykkesscenarier på et digitalt spillbord (som et dataspill) for å få et mer realistisk perspektiv på denne opplæringen. Dette skal gjennomføres ved å lage et program basert på dagens 3D First Person Shooter (FPS) spill, og som bruker samme terminologi som et 3D FPS spill.

Det ble holdt et forprosjekt i faget "Objektorientert Systemutvikling" høsten 2003 (Prosjekt Smalvik). Dette kan sees på som første avgrenset utkast til løsningen, og var vårt utgangspunkt da vi begynte å løse denne oppgaven.

1.2 Definisjon av oppgaven

Vårt oppdrag har vært å lage en Pilot/prototype av dette digitale spillebordet, med minst ett fungerende scenario. Det skal kunne «spilles» av en person og styres av en kontrollør.

Denne modellen skal rendres av en ferdiglaget 3D-motor. Det skal også lages «verktøybokser», med hendelser og verktøy. Disse hendelsene skal kunne settes ut etter hvert som spillet forløper, og vil være reaksjoner på valg tatt av spilleren, f.eks. sette ut sperringer, «parkere» utrykningskjøretøy osv.

Scenariet skal være et tredimensjonalt miljø med landskap, bygninger, gjenstander (for eksempel biler, sperrebukker, steiner, trær og lignende), mennesker, hendelser, klimatiske faktorer (vær), samt diverse effekter i forbindelse med brann, røyk, vann osv.
Både spiller og kontrollør skal kunne foreta valg som påvirker hva som skjer videre i scenariet. F.eks. skal kontrollør kunne sette ut «Venteplass for Ambulanser» og lignende. Ambulansene skal da kjøre dit.

Vår hensikt er å utvikle et system hvor en server kjører scenariet og deltagerne kobler seg opp på serveren via arbeidsstasjoner og "spiller" via en lokal nettverksforbindelse. Siden prototypen skal kunne brukes som et utgangspunkt for videre utvikling , har vi som mål å lage en grundig dokumentasjon og skrive kildekode på en måte som gjør gjenbruk enkelt.

1.3 Målgruppe for rapporten

I utgangspunktet er det tre hovedmottagere av denne rapporten. Disse tre er veilederen for hovedprosjektgruppen vår, ekstern sensor for hovedprosjektet og oppdragsgiver, i vårt tilfelle Sivilforsvaret. Andre mottagere er studenter ved HiG som skal gjøre et eget hovedprosjekt og personer som på vegne av sivilforsvaret skal bygge videre på vårt resultat.

Hensikten med denne rapporten er å gi en grundig beskrivelse av hvordan vi jobbet oss gjennom dette prosjektet og leseren av denne rapporten skal få innsikt i hvordan vi kom frem til et resultat. Veileder og sensor skal bruke denne rapporten når de skal bedømme resultatet vårt og da er det viktig for oss å få frem hva vi har gjort, hvilke avgjørelser og vurderinger vi gjorde underveis, vår vurdering av resultatet og hvordan vi har jobbet som gruppe. Oppdragsgiver skal bruke rapporten som en dokumentasjon og beskrivelse for produktet vi har utviklet for dem.

1.4 Formål med prosjektet

Sivilforsvarets ønske er å kunne kjøre ulykkesscenarier på et digitalt spillbord (som et dataspill) for å få et mer realistisk perspektiv på et scenario under opplæring av beredskapspersonell, enn hva som oppnås med det eksisterende spillbordet som blir brukt i dag. Det eksisterende spillbordet har en rekke praktiske ulemper med hensyn på størrelse, slitasjeskader og transport. Med digitalt spillbord ville man eliminert disse problemene, det vil dessuten være lettere å legge til nye scenarier. Et digitalt spillbord muliggjør desentralisert simulering over større avstander, ved bruk av klienter og en server. Deltagerne kan altså være geografisk spredt, og allikevel få utnyttet spillbordets pedagogiske effekt.

Effektmål:
Sivilforsvarets mål med denne oppgaven er å få utviklet en pilot som kan brukes til å demonstrere hva som er mulig å få til med den teknologien som eksisterer i dag. Prototypen skal ta for seg et forhåndsbestemt scenario og det skal være mulig å ”spille” på dette. Denne prototypen skal kunne brukes som et grunnlag for videre utvikling seinere.

1.5 Rammer

Avgrensinger må foretas på flere steder. Siden oppgaven skal løses som et «3D-spill», vil dette stille høye krav til hardware. Siden sivilforsvaret har en maskinpark som ikke er beregnet for slik bruk må prototypen vår være effektiv og ha god ytelse på deres maskiner. Siden vi kun skal utvikle en prototype har vi ikke lagt vekt på å utvikle detaljert grafikk. Vi har fokusert mer på å utvikle elementer og funksjonalitet som er nødvendig å ha med i et ulykkesscenario.

1.6 Gruppas arbeidsformer

Arbeidsformen på gruppa har vært slik at alle har jobbet med hver sin oppgave, f.eks. har vi programmert på hver våre ting. Vi har fordelt arbeidet slik på grunn av at vi har for liten tid til at alle kan være med å gjøre alt og noen medlemmer av gruppen har jobbet hjemmefra på grunn av mangel på utstyr på skolen. Vi har samarbeidet om feilsøking og testing av koden vår underveis i prosjektet.
Utstyr vi har brukt på skolen er tre stasjonære datamaskiner med Windows, to private og en utdelt av skolens IT-tjeneste. I tillegg har vi brukt en Linux-maskin som server. På denne serveren lagrer vi all kildekode og dokumenter, styrt vha. konfigurasjonsstyring (CVS), slik at vi kan jobbe med prosjektet hjemmefra.
Gjennom hele prosjektet har vi hatt faste møter med veileder en dag i uken. På disse møtene har vi fått råd og tips om hvordan vi skulle arbeide med de forskjellige tingene. Internt i gruppa har vi hatt møter etter behov og har brukt fremdriftsplanen og lister med oppgaver for å styre fremgangen i arbeidet. Gjennom hele prosjektet har vi skrevet statusrapporter, møtereferater og logger.
Vi har hatt flere samtaler med oppdragsgiver, Mads Ringen, samt korrespondanse per e-post. I tillegg har vi vært på Starum et par ganger og sett på modellbordet de har der. Første gang bare for å se hvordan det så ut og for å få litt følelse for oppgaven, senere for å se det demonstrert under en simulering av en ulykke. Uka før påske var det en reell simulering av en ulykke. Vi var der og observerte hvordan dette foregikk, for å få noen ideer til vårt prosjekt. På denne dagen viste vi også frem en prototype av applikasjonen vår.

1.7 Egen bakgrunn og kompetanse

Gruppens medlemmer består av studenter fra dataingeniørlinjen på Høgskolen i Gjøvik, herunder studieretningene drift og programmering. Gruppen har derfor kompetanse innen programutvikling, systemutvikling og drift av server.

1.8 Teknologi

Prosjektet tar i bruk flere teknologier som Genesis3D (generering av 3D grafikk), C/C++ programmering, 3D-modellering og konfigurasjonsstyring. Genesis3D er en grafikkmotor og var helt ukjent for gruppen før prosjektstart. Gruppens medlemmer måtte derfor sette seg inn i denne teknologien tidlig i prosjektfasen. Genesis3D er utviklet med Microsoft Visual Studio .Net og programmert i C/C++ . Til konfigurasjonsstyring ble CVS benyttet, som ingen i gruppen hadde erfaring med. Jfr. ansvarsområdene innad i gruppen, ble kunnskap tilegnet til å sette opp og drifte en CVS-server. Alle i gruppen måtte dessuten lære seg bruk av CVS til konfigurasjonsstyring.

1.9 Øvrige roller

Veileder:
Vår veileder gjennom dette prosjektet har vært Tom Røise. Røise er lærer ved HiG og underviser i systemutvikling og objektorientert systemutvikling. Hans rolle har vært å gi oss tips, råd og veiledning i gjennom hele prosjektet.

Oppdragsgiver:
Vår oppdragsgiver er Sivilforsvaret Starum. Vår kontaktperson i Sivilforsvaret, Mads Ringen, har vært med å utforme alle krav til systemet og hvordan løsningen bør se ut. Vi har også fått en gjennomgang av hvordan de driver opplæring og sett modellbordene deres i bruk. Dette har gitt oss verdifull informasjon som vi har brukt under utvikling av systemet.

Prosjektgruppe på datalinjen:
Før vi valgte denne oppgaven ble det gjort en vurdering av dette prosjektet av en gruppe studenter med fordypning i systemutvikling i 3. klasse på datalinjen ved HiG. Denne gruppen hadde som oppgave å evaluere muligheten for å lage et virtuelt spillbord for sivilforsvaret. Rapporten skulle presentere en teoretisk prototype og anbefalinger til hvordan problemet burde løses men skulle ikke bringe frem noen konkret løsning av problemet. I rapporten er det gjort en del analyser av problemer angående krav til systemet, design, implementering og valg av grafikkmotor. Rapporten inneholder forskjellige forslag til hvordan disse tingene skal løses.
Vi studerte denne rapporten og brukte den under vår vurdering av hvordan vi skulle løse oppgaven. Vi har ikke konkret brukt noen av løsningsforslagene i denne rapporten, men vi har studert anbefalinger og analyser da vi utarbeidet vår egen plan for prosjektet.
Rapporten som denne gruppen produserte er lagt ved på CD.

1.10 Organisering av rapporten

Rapporten er delt inn i 11 kapitler:

Kapittel 1 – Innledning
Gir en kort innføring i prosjektet og prosjektarbeidet.

Kapittel 2 – Kravspesifikasjon
Kravspesifikasjonen er tilpasset rapporten. Den beskriver alle krav til systemet som ble definert på forhånd.

Kapittel 3 – Design
Dette kapitlet gir en grundig beskrivelse av systemet og beskriver hvordan vi har løst oppgaven.

Kapittel 4 – Implementering, koding og produksjon
Gir en beskrivelse av alle utviklingverktøy som er benyttet i prosjektet, gir detaljerte beskrivelser av hvordan vi utviklet noen av de forskjellige inkrementene i prosjektet og viser kode-eksempler.

Kapittel 5 – Testing og kvalitetssikring
Beskriver hvordan testing har foregått og hvilke strateier som er brukt.

Kapittel 6 – Installasjon/realisering
Beskriver hvordan sluttproduktet er bygd opp og hvilke eksterne ressurser som behøves for å bruke applikasjonen.

Kapittel 7 – Beskrivelse av prosessen
Begrunnelse på hvorfor vi har valgt å bruke Inkrementell systemutvikling. Beskrivelse av de forskjellge fasene i denne modellen.

Kapittel 8 – Diskusjon av resultater
Diskusjon og drøfting rundt produktet, arbeidsrutiner og valg som ble gjort underveis.

Kapittel 9 – Konklusjon
De slutninger som vi kom frem til i forrige kapittel.

Kapittel 10 – Litteraturliste
Hvilke kilder vi har brukt under prosjektet og en ordliste over tekniske ord og uttrykk som er brukt i rapporten.

Kapittel 11 – Vedlegg
Vedleggene består av dokumentasjon av kode, møtereferater, fremdriftsplaner, logger, fullstendig Designdokument og annen dokumentasjon.

1.11 Terminologiliste

Her følger en liste over tekniske ord og uttrykk som er brukt i rapporten.

- 1. og 3. persons view: Synsperspektivet til spilleren. Førstepersons-view: Når man ser "gjennom øynene" på personen man spiller. Tredjepersonspersons-view: Når man ser ting bakfra, ovenifra eller fra siden, på selve spilleren.
- 2D: To-dimensjonalt.
- 3D: Tre-dimensjonalt.
- 3D-modellering: Tegning og forming av tre-dimensjonale figurer og objekter som brukes i et 3D miljø.
- 3D-spill: Et dataspill som bruker 3D-grafikk.
- 3DStudio Max: Dette er et anerkjent 3D-modelleringsprogram som vi brukte for å lage 3-dimensjonale modeller til bruk i spillet.
- Actor Creator: Dette er et program vi brukte for å kompilere .nfo(3d-modellen) og .key (Animasjon) filer til et .act format.
- AI: Artificial Intelligence, kunstig intelligens/tankemønster til Kunstige Aktører.
- Aktør: Publikum og de forskjellige etaters deltagere, f.eks. brannmenn og politimenn.
- API: Application Programming Interface. En API er et verktøy som brukes til å utvikle programvare. I vårt tilfelle så er det en grafikkmotor som inneholder alle funksjoner og elementer man trenger for å genere 3d-grafikk.
- Bones: Dette er en del av et skjelett i en 3-dimensjonal modell. Og fungerer på samme måte som et bein i vår egen kropp.
- Console: Kommandolinje hvor bruker kan skrive kommandoer til Dirus.
- CVS: (Concurrent Versions System) Verktøy for konfigurasjonsstyring.
- Dedicated server: Server som kun tar seg av en oppgave og ikke kan brukes til noe annet så lenge den tar seg av denne oppgave. Brukes ofte til å kjøre spill med mange klienter koblet til.
- DirectX: Programvare fra microsoft som brukes i windows for håndtering av grafikk. Denne blir stadig oppdatert og man holder styr på dette via versons-nummer, f.eks. DirectX 9.0a.
- Entitet: Brukt i forbindelse med kart og kart-editor. En entitet er et objekt i kartet, for eksempel røyk, brann, spiller osv.
- FPS: First Person Shooter, på norsk Første Persons Skytespill. Vil si at perspektivet ditt er gjennom øynene til personen i spillet.
- Full Duplex: At kommunikasjon mellom to eller flere parter kan foregå samtidig i begge retninger.

- Genesis3D: Genesis3D er navnet på grafikkmotoren vi bruker i prosjektet vårt. Genesis3D er en 3D-grafikkmotor som er utviklet av Eclipse. For mer info se www.genesis3d.com.
- Grafikkmotor: En grafikkmotor er en software man bruker for å generere grafikk som vises på en pc skjerm. Det finnes mange forskjellige grafikkmotorer til forskjellige formål. Grafikkmotorer er biblioteker med funksjoner, datastruktur og objekter som programmer, slik som spill, bruker for å generere grafikk.
- GUI: Står for Graphical User Interface eller grafisk brukergrensesnitt på norsk. En GUI er det brukeren av en datamaskin bruker til å kommunisere med systemet, f.eks. ved hjelp av vinduer og ikoner som man kan klikke på.
- GTest: Er et førstepersons skytespill som vi har tatt utgangspunkt i og videreutviklet.
- HUD: «Heads-Up-Display,» tekst og annen informasjon som vises i selve synsfeltet til spilleren.
- Klassediagram: Grafisk fremstilling av hvilke klasser man har i systemet og hvordan de henger sammen.
- Klasser: Beskrivelse av hva et objekt i systemet skal inneholde og hva det skal kommunisere med. Man kan opprette mange objekter av en klasse.
- Klient: Datamaskin som logger seg på en server og benytter seg av den informasjonen som ligger der.
- Kontrollør: Brukeren som har rollen som kontrollør skal ha oversikten over alt som skjer og skal veilede skadestedsleder.
- Kunstige Aktører (KA): Aktører i spillet hvor adferd styres i sin helhet av Dirus, for eksempel publikum og diverse personell.
- LAN: Local Area Network. Et lokalt nettverk med datamaskiner som er uavhengig av internett for å kommunisere med hverandre.
- Maskinkraft: Hvor kraftig en datamaskin er. Man tenker her på prosessorkraft, minne og evne til å håndtere grafikk.
- MaxScript: Dette er et skript skrevet for å utføre en rekke operasjoner i 3D Studio Max. I vårt tilfelle for å eksportere 3d-modeller til et .nfo filformat.
- Mesh: En samling av punkter som tilsammen utgjør et nett og fremstiller et objekt i tredimensjonal form.
- Modellbord: Opprinnelig 5 moduler à 120x240cm i målestokk 1:87 som utgjør det nåværende simulerings-bordet.
- Modelleringsverktøy: Verktøy til utvikling av tre-dimensjonale modeller.
- Modus: Tilstand en datamaskin er i, f.eks. klient-modus eller server-modus.
- Open Source: Betyr at kildekoden til produktet eller applikasjonen er fri for alle til å bruke eller endre

- Rendring: Når noe blir rendret så betyr det at det blir tegnet opp på skjermen. Når man rendrer en animasjonsfilm så tegnes hvert bilde opp og settes i en sekvens som avspilles. En grafikkmotor rendrer grafikk fortløpende mens ting skjer, i motsetning til en animasjonsfilm som blir rendret ferdig før den vises på skjermen.
- Scenario: I denne sammenheng er et scenario et virtuelt miljø hvor det blir iscenesatt hendelser som brukeren skal forholde seg til. Et scenario består typisk av et utendørs landskap med hus, trær og busker. Dette miljøet bruker vi til å i scenesette en ulykke, f.eks. en brann.
- Server: Datamaskin som inneholder informasjon som den deler mellom klienter på et nettverk og/eller tar seg av kommunikasjon mellom klienter.
- SFSKS: Sivilforsvarets skole på Starum.
- Skadestedsleder: Brukeren som har rollen som skadestedsleder skal forholde seg til det som skjer og ta avgjørelser og utføre handlinger.
- Spillbarhet: Betegnelse på hvor mye brukeren/spilleren av denne demo'en kan gjøre i scenariet. Man tenker da på om det finnes mange valgmuligheter og hvordan de valg som blir tatt påvirker spillopplevelsen, hvordan man kommuniserer med objekter og medspillere og hvor lett det er å bruke de mulighetene som finnes.
- Spillebord: Se Modellbord
- TCP/IP protokollen: Dett med regler for hvordan datamaskiner skal kommunisere med hverandre over internett eller et lokalnettverk. TCP/IP er en internasjonal standard som brukes av de fleste operativsystemer og platformer i dag.
- Use Case: Grafisk fremstilling av hvilke rettigheter de forskjellige bruker-rollene av systemet har og hvilke funksjoner de har tilgang til.
- Utviklingsmiljø: Hvilken applikasjon (eller samling av) som blir benyttet til programmering og kompilering av kildekode.
- Walk-cycle: En animasjon som starter og slutter likt, så hvis den blir spilt av i en løkke oppfattes det som en kontinuerlig bevegelse.

2 Kravspesifikasjon

Kravspesifikasjonen ble skrevet før vi begynte å arbeide på prototypen og ble utarbeidet i samarbeid med oppdragsgiver. Dette kapitlet inneholder alle krav vi hadde til systemet før vi startet med utviklingen.

2.1 Introduksjon

2.1.1 Bakgrunn

Sivilforsvarets skole på Starum driver med opplæring av beredskapspersonell. Hovedsakelig utdanner de skadestedsledere hvor de tar for seg forskjellige scenario, f.eks: Bilulykker, branner, rasulykker, tunnelulykker, jernbaneulykker, farlig gods, industriulykker o.l. I dag bruker de et fysisk modellbord på 15 kvadratmeter. Dette modellbordet er dyrt å transportere og vedlikeholde, og av den grunn hadde det vært hensiktsmessig å bruke et virtuelt scenario istedet for et fysisk modellbord. Sivilforsvarets mål med denne oppgaven er å få en demonstrasjon på hvordan dette kan løses.

2.1.2 Kort om krav til systemet

Prototypen skal kjøre et ulykkesscenario. Skadestedsleder skal ha en førstepersons-«view» og skal kunne foreta valg og styre mannskapene på ulykkesstedet. En kontrollør skal ha et oversiktsbilde av hva som skjer, og han/hun skal kunne påvirke hendelser i scenariet. Scenariet skal kjøres på en server, og de som skal "spille" logger seg på serveren via arbeidsstasjoner. Grafikken blir simpel, med lite detaljer og avanserte hendelser. Det skal legges vekt på funksjonalitet og brukervennlighet.

2.2 Generelle krav

2.2.1 Forskjellige modus

Prototypen skal ha mulighet for å velge om den skal fungere som server, eller koble seg til en allerede eksisterende server. Det skal gå an å koble seg til en server via LAN eller Internett.

I server-modus:

- Skal ha kontroll over fremdriften av scenarioet
- Skal ta seg av kommunikasjon med klientene (via TCP/IP-protokollen).

I klient-modus:

- Skadestedsleders GUI skal være et førstepersons-«view». Det skal også være mulighet for å velge verktøy via en meny eller HUD.
- Spilleleder/kontrollør skal ha mulighet til å se oversiktsbilde av hele scenariet og bør kunne å styre noen hendelser. En mulighet for utvidelse er å gi spilleleder/kontrollør mulighet til å klikke seg inn på en enkelt spiller og se det den ser fra dens perspektiv.

2.2.2 GUI

GUI-en skal ha en meny med diverse valg:

- Starte server, eller koble til eksisterende server
- Velge om en vil være spiller eller kontrollør
- Om en skal koble til med et lagret spill, eller starte som ny spiller/spilleleder
- Bestemme hvilke taster som skal styre forskjellige funksjoner og bevegelser i spillet.
- Velge innstillinger for lyd og grafikk
- Velge om man vil avslutte programmet.

2.2.3 Scenario

Vi har i sammen med oppdragsgiver bestemt oss for i utgangspunktet å utvikle ett scenario. Siden dette kun skal være en demonstrasjon så trenger vi bare ett scenario.
Hvis vi får tid til det så kan det hende vi utvikler et helt nytt scenario i tillegg, eller lager to versjoner med det samme landskapet.

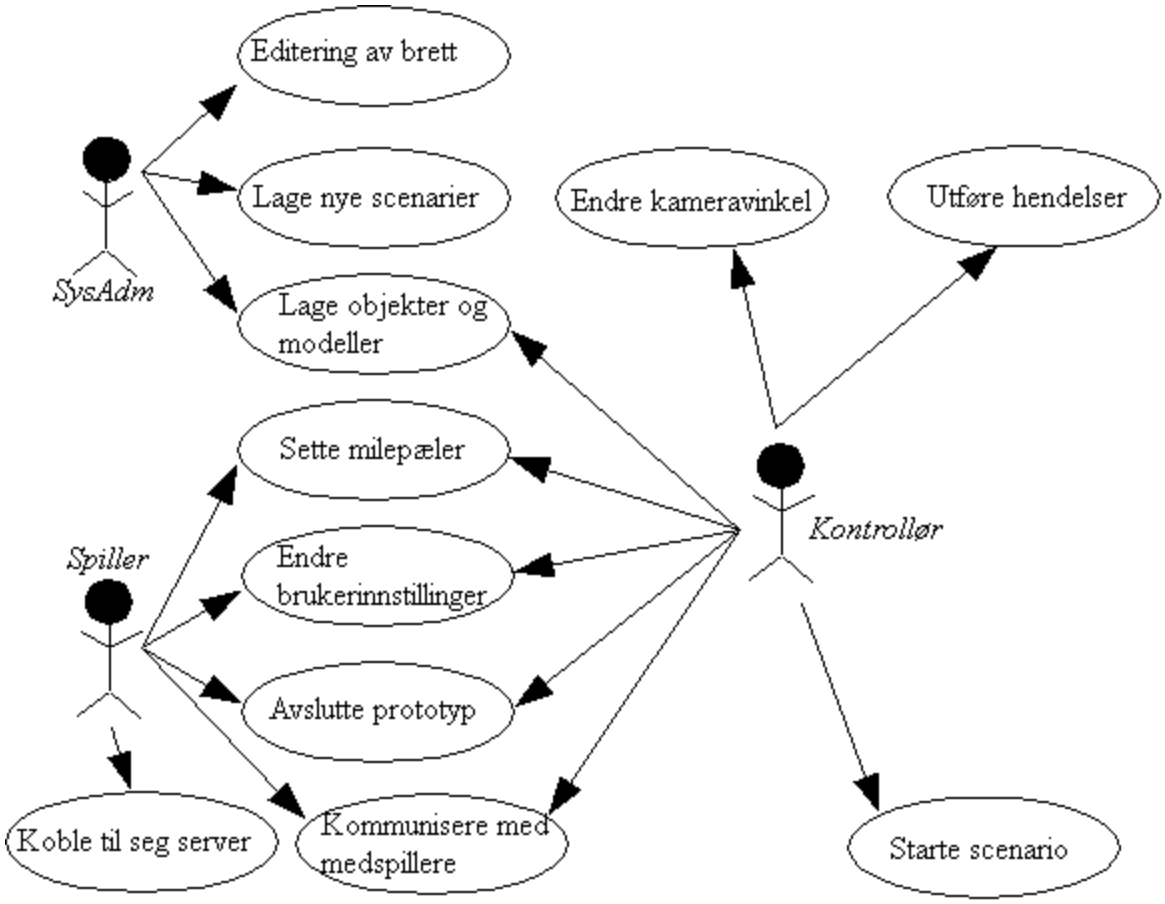
- Scenarioet skal være en eksplosjonsartet brann i et industribygg.
- Rundt industribygget skal det være flere bygninger og objekter (f.eks. Trær, tønner, containere o.l.)
- Det skal være bevegelige mennesker og kjøretøy.
- Spesielle grafiske elementer som branner og røyk.
- Vær og vind. F.eks. kan vindretning ha noe å si for brannutviklingen

2.3 Krav til funksjonalitet og Use Case

Det stilles en del krav til funksjonaliteten i denne prototypen. Dette skal føre til at et scenario oppleves så realistisk som mulig. Disse kravene er:

- Spilleren skal være faglig leder på skadestedet.
- Spilleren skal ha mulighet til flytte kjøretøy og sette opp sperringer på hvilket som helst tidspunkt i scenariet.
- Spiller skal ha mulighet for å plassere ut milepæler:
  - For å kunne gå tilbake til et tidligere punkt i spillet
  - For å kunne avslutte spillet, og gå starte opp igjen på et senere tidspunkt.
- Når en spiller kommer i kontakt med en «statist» i spillet bør spilleren kunne velge om han/hun vil snakke med vedkommende.
- Spilleren skal kunne bevege seg rundt i scenariet som han vil.
- Spilleren skal bestemme hvordan problemer skal løses.
- Spilleleder skal kunne styre noen hendelser i scenariet, f.eks. sette inn eksplosjoner og starte branner.
- Spilleleder skal ha mulighet til å stoppe spillet (pause-funksjon) for å kunne lagre eller fortsette spillet.
- Spilleleder skal ha mulighet til å spille inn opptak av spillet, for å kunne bruke som demonstrasjon ved senere anledning.

2.3.1 Use Case



Figur 1UseCase

2.3.2 Use Case beskrivelse

Lage brett:
Aktør: System administrator.
Type: Konfigurering.
Beskrivelse: Det skal lages en topologi, enten et utendørs landskap eller et innendørs miljø. Man skal sette inn objekter (steiner, trær, hus, kjøretøy osv) som er utviklet på forhånd, enten med samme program eller med et annet program. Deretter skal man kunne sette inn dynamiske elementer slik som branner, røyk og vær. Til slutt skal man legge inn hendelser og «triggere.»
Betingelser: Må bruke verktøy som er tilpasset grafikkmotor. For å lage terreng kan man bruke Genesis World Editor (GEdit). Man kan også bruke 3DStudio Max til å lage modeller og importere disse inn i Gedit.

Lage objekter og modeller:
Aktør: System administrator.
Type: Konfigurering.
Beskrivelse: Her skal man kunne modellere figurene i 3DStudio Max for så og konvertere og importere dem inn i Gedit, slik at man kan putte dem inn i et brett/scenario. Hvis en modell skal ha bevegelser, så man bør også animere modellene i 3DStudio Max og importere disse bevegelsene inn i Genesis motoren.
Betingelser: Må bruke verktøy som er tilpasset grafikkmotor. 3DStudio Max og Genesis Actor Studio kan brukes til å lage modeller.

Editering av brett:
Aktør: System administrator.
Type: Konfigurering.
Beskrivelse: Her må man gjøre det samme som når man lager et brett, bortsett fra at man tar utgangspunkt i en topologi og et scenario som allerede eksisterer. Man skal kunne legge til eller fjerne objekter eller «triggere» og/eller forandrer på skript.
Betingelser: Må bruke verktøy som er tilpasset grafikkmotor. For å lage terreng kan man bruke Genesis World Editor (GEdit). Man kan også bruke 3DStudio Max til å lage modeller og importere disse inn i Gedit.

Forandre brukerinnstillinger:
Aktører: Kontrollør og Spiller.
Type: Konfigurering.
Beskrivelse: Man skal kunne navigere via en meny slik at man kan forandre diverse innstillinger, slik som brukernavn, skjermoppløsning, lydvolum og konfigurering av tastatur.
Betingelser: -

Starte scenario:
Aktør: Kontrollør.
Type: Handling i prototype.
Beskrivelse: Kontrolløren skal via en meny kunne starte et scenario. Når dette scenariet startes så skal prototypen gå i servermodus og fungere som en server.
Betingelser: Kun kontrollør har denne menyen.

Koble til server:
Aktør: Skadestedsleder.
Type: Handling i prototype.
Beskrivelse: Skadestedsleder skal via en meny kunne koble seg til en server som en kontrollør har startet.
Betingelser: Kun Skadestedsleder har tilgang denne menyen. Det må være startet en server av en kontrollør på lokalnettverket eller på Internett.

Sette milepæler:
Aktører: Kontrollør og Spiller.
Type: Handling i prototype.
Beskrivelse: Det skal være mulig å sette milepæler og lagre tilstanden i spillet på forskjellige tidspunkt, slik at man kan gå tilbake til en milepæl og prøve på nytt hvis man mislykkes med noe. For å få til dette må alle variable lagres for hver milepæl. I første omgang er de kontrollør som skal ha denne muligheten, men spiller bør også kunne gjøre dette.
Betingelser: save-filen må lagres på både kontrollør og spiller sine maskiner.

Kommunikasjon med medspiller:
Aktører: Kontrollør og Spiller.
Type: Handling i prototype.
Beskrivelse: En måte å kommunisere i spillet på, f.eks. chat eller stemme-kommunikasjon. For å skrive inne en melding eller snakke via mikrofon så skal man kunne trykke på en bestemt tast, deretter skrive inn meldingen eller snakke i mikrofonen. Alle meldinger vil komme opp som tekst et bestemt sted på skjermen eller ut gjennom høyttalerne.
Betingelser: Hvis man skal ha stemmekommunikasjon må man ha høyttalere og mikrofon.

Avslutte prototyp:
Aktører: Kontrollør og Spiller.
Type: Handling i prototype
Beskrivelse: Via navigering av en meny skal brukeren ha mulighet til å avslutte prototypen og returnere til Windows.
Betingelser: -

Endre kameravinkel:
Aktør: Kontrollør.
Type: Handling i prototype.
Beskrivelse: Mulighet til å endre kameravinkel midt i scenariet for å få full oversikt. Vi ser for oss at kontrollør skal ha oversikten over hele scenariet fra oven, men at han/skal også skal kunne bevege seg fritt i scenariet for å kunne se alt fra alle vinkler. Kontrolløren bør også ha mulighet til å se ting "gjennom øynene" til den som spiller(bytte mellom første- og tredjeperson). Kontrolløren skal ikke ha noen fysisk form han/hun skal bare observere.
Betingelser: Må være uavhengig av en "person" i spillet.

Utføre hendelser:
Aktør: Kontrollør.
Type: Handling i prototype.
Beskrivelse: Kontrolløren skal ha mulighet til å forandre scenariet "realtime", dvs at det skal være mulig å legge inn eksplosjoner, flammer, røyk, objekter inn i scenariet i sanntid mens spilleren spilleren spiller. Dette skal kunne gjøre via en meny eller lignende.
Betingelser: Hvis man skal legge til objekter og hendelser må disse finnes på alle maskinene som er logget på serveren.

2.4 Supplementær kravspesifikasjon

2.4.1 Maskinkrav og andre krav

- Prototypen skal kunne kjøres på datamaskiner med 500MHz og med skjermkort med 3D akselerasjon.
- Grafikkmotoren krever DirectX 6.0 eller nyere, så det må inkluderes i en installasjonen av programmet.
- Det skal være mulighet for å spille på storskjerm.
- Det vil være mulighet for at minst 2 brukere kan logge seg på serveren å «spille» likt, men vi prioriterer ikke mulighet for mange spillere i første omgang.

2.4.2 Operasjon i feilsituasjoner

- Feil under eksekvering av programmet før logges enten til ei fil på den maskinen som programmet kjøres på eller logges på en server.
- Hvis det ikke er feil som er kritiske for om programmet kjøres eller ei, bør ikke disse vises til bruker.

2.4.3 Krav til installasjon

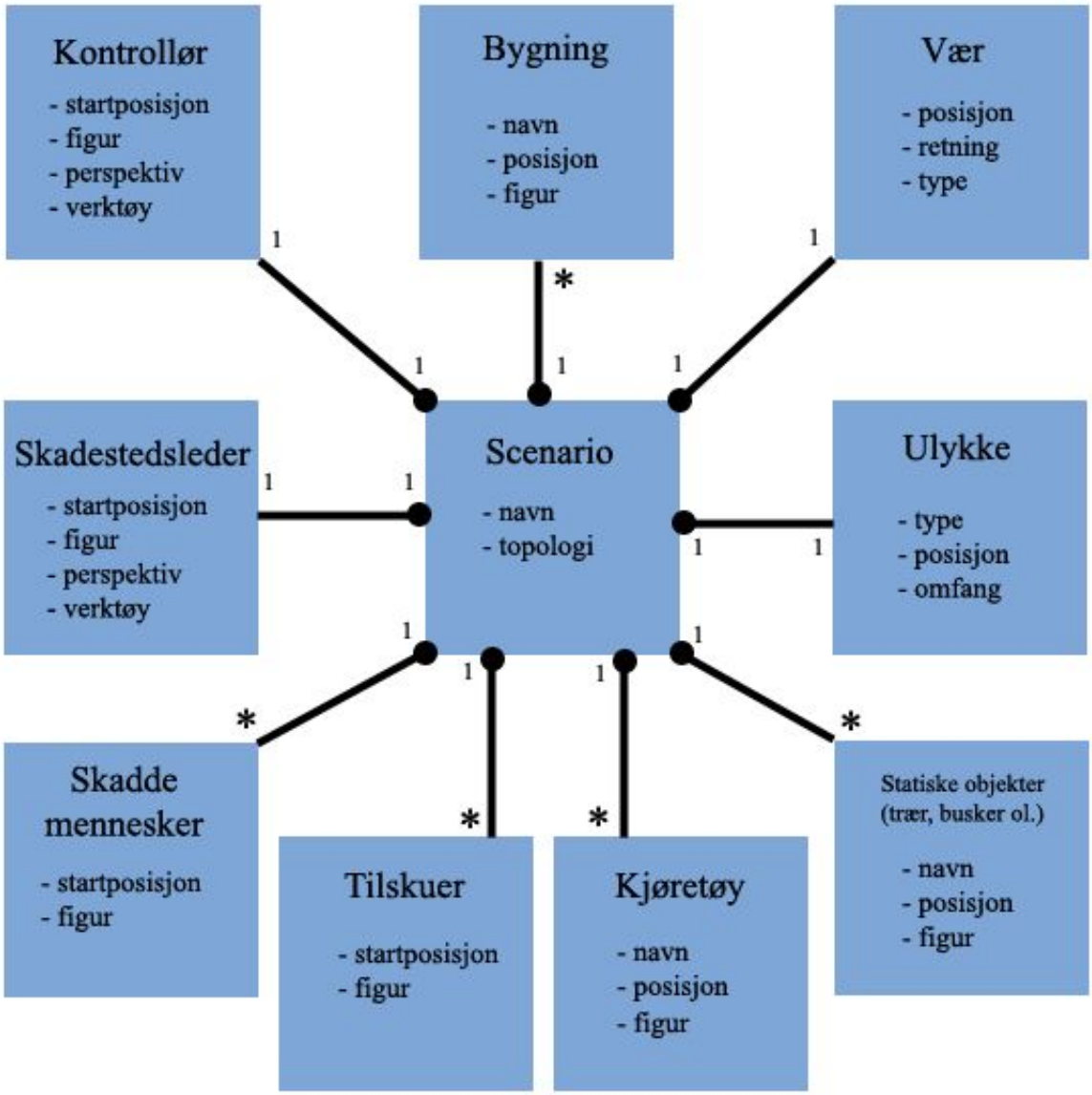
Installasjon av prototypen skal gjøres så enkel som mulig slik at det stilles minst mulig krav til brukeren. Det skal være en installasjonsfil eller alle filer og ressurser som trengs for å kjøre prototypen skal finnes i en mappe, slik at man bare kan kopiere denne fra en CD.

2.4.4 Mulighet for videre utvikling

Det skal være mulig å fortsette å videreutvikle denne prototypen. Når prosjektet er ferdig og levert skal det finnes dokumentasjon på arbeidet vårt og det bør formuleres en del anbefalinger til forbedringer og nye løsninger.

2.5 Konseptuelt klassediagram og beskrivelse

Det konseptuelle klassediagramet er en grafisk oversikt over hva ulykkesscenariet skal inneholde av objekter og aktører. Hver av de blå firkantene er en klasse. Hver klasse har et navn og en liste med attributter under navnet. Hovedklassen er scenariet. Alle de andre klassene er elementer som skal inn i et scenario. Tallene som vises i diagrammet forteller noe om antallet objekter av de forskjellige klassene i scenariet. Det kan bare være en kontrollør og en skadestedsleder i et scenario, mens det kan være mange tilskuere, kjøretøy og skadde personer.



Figur 2Konseptuelt klassediagram

2.5.1 Beskrivelse av det konseptuelle klassediagrammet.

Her følger en beskrivelse av de enkelte klassene i klassediagramet.

Scenario: Et scenario skal i utgangspunktet inneholde omgivelsene og topologien skal brukes i ulykkesscenariet. Klassen Scenario har to attributter:
Navn: navnet på scenariet.
Topologi: Topologien vi bestå av veier, gress, jord o.l.
Objekter av alle de andre klassene skal opprettes inne i disse omgivelsene og handlingen i ulykkesscenariet skal foregå her.

Vær: Det skal være en eller annen form for vær inne i scenariet som skadestedsleder skal ta hensyn til. Klassen Vær har tre attributter:
Posisjon: Været må plasseres ut i scenariet på et passende sted, slik at det påvirker scenariet på en realistisk måte.
Type: Det skal være flere typer vær, f.eks. regnvær, snø og vind.
Retning: Det skal gå an å styre retningen på været, f.eks. vindretningen.

Ulykke: Det skal være én ulykke i scenariet. Vi har i sammen med oppdragsgiver kommet frem til at dette skal være en brann i en bygning, men det skal også være støtte for andre typer ulykker. Klassen ulykke har tre attributter:
Type: hvilken type ulykke det er. f.eks. brann, bilulykke, eksplosjon o.l.
Posisjon: posisjonen til ulykken i scenariet, f.eks. i en bygning eller ved en vei.
Omfang: omfanget av ulykken. f.eks. om brannen skal spre seg.

Bygning: Scenariet skal inneholde flere bygninger og hus. Det skal være en fabrikk som det skal være en brann i, og det skal være noen bygninger rundt denne fabrikken.
Klassen bygning har tre attributter:
Navn: navnet på bygningen.
Posisjon: posisjonen til bygningen i scenariet.
Figur: figuren til bygningen

Kjøretøy: Scenariet skal inneholde flere kjøretøy. Disse skal enten være privatbiler eller offentlige kjøretøy, slik som brannbiler, politibiler, sykebiler o.l.
Klassen kjøretøy har tre attributter:
Navn: Navnet på kjøretøyet, f.eks. brannbil1.
Posisjon: Posisjonen hvor kjøretøyet blir utplassert i scenariet.
Figur: Figuren til kjøretøyet.

Statiske objekter: I tillegg til bygninger og biler skal det plasseres ut forskjellige statiske objekter, dvs. objekter som ikke beveger seg i scenariet. Slike objekter kan være trær, busker, steiner o.l. Klassen statiske objekter har tre attributter:
Navn: Navnet på objektet.
Posisjon: Posisjonen hvor objektet blir utplassert i scenariet.
Figur: Figuren til objektet (f.eks. et tre).

Kontrollør: Det skal være én kontrollør i scenariet. Denne brukeren skal være usynlig for skadestedsleder slik at scenariet blir mest mulig realistisk. Kontrolløren skal kunne bevege seg over hele scenariet og bruke sine verktøy. Denne klassen har fire attributter:
Startposisjon: Posisjonen i scenariet hvor kontrolløren starter.
Figur: Figuren til kontrolløren, denne skal være usynlig.
Perspektiv: Brukeren skal kunne velge mellom flere perspektiver, slik som førsteperson eller oversikts-perspektiv.
Verktøy: Brukeren skal kunne velge mellom flere forskjellige verktøy, som f.eks. sperrebukker o.l.

Skadestedsleder: Det skal være én skadestedsleder i scenariet. Denne brukeren skal kunne bevege seg over hele scenariet og bruke sine verktøy.
Denne klassen har fire attributter:
Startposisjon: Posisjonen i scenariet hvor skadestedsleder starter.
Figur: Figuren til skadestedsleder. Dette skal være en menneskefigur med en vest på overkroppen, tilsvarende den sivilforsvaret bruker i virkeligheten.
Perspektiv: Skadestedsleder skal kunne velge mellom flere perspektiver, slik som førstepersons- eller oversiktsperspektiv.
Verktøy: Brukeren skal kunne velge mellom flere forskjellige verktøy, som f.eks. flagg o.l.

Skadde mennesker: Scenariet skal inneholde noen kunstige aktører som skal være skadde. Disse skal plasseres rundt om i omgivelsene for å gjøre scenariet realistisk.
Denne klassen har to attributter:
Startposisjon: posisjonen i scenariet hvor det skadde mennesket starter.
Figur: figuren til dette skadde mennesket.

Tilskuer: For å gjøre scenariet mer realistisk så skal det plasseres ut tilskuere på forskjellige steder i scenariet. Disse skal bevege seg rundt og må evakueres til et punkt som skadestedsleder har definert.
Denne klassen har to attributter:
Startposisjon: posisjonen i scenariet hvor tilskueren starter.
Figur: figuren til tilskueren.

2.6 Oppsummering

I utviklingsfasen i dette prosjektet begynte vi med å utvikle et brett og lage modeller. Vi valgte å begynne med dette fordi vi allerede viste hvordan vi skulle gjøre dette og det var en tidkrevende prosess. Parallelt med dette studerte vi kildekoden til motoren og fant ut hvor vi måtte gjøre forandringer, eventuelt om vi kunne finne ferdige moduler, utviklet av andre, som vi kunne bruke direkte eller med modifikasjoner. Når vi fikk på plass store deler av topologi og modeller begynte vi å utvikle en GUI. Denne GUI-en består av en hovedmeny, undermenyer, diverse 2D-grafikk (statusbar) som ble plassert i forgrunnen på skjermbildet for å vise diverse informasjon. Når denne GUI-en er på plass må vi legge til funksjonalitet til menyene, f.eks. slik at det går an å logge på og av en server. Når vi har fått på plass det mest nødvendige skal vi utvikle hendelser til scenariet. Dette er en viktig bit som vi absolutt bør få noe til å fungere. Oppdragsgiver har presisert at det viktigste for dem er å få en prototype med et scenario hvor det skjer ting og hvor man kan foreta valg som påvirker det som skjer, grafikk og utseende er ikke så viktig. En dialog mellom spiller og kontrollør er viktig, og av den grunn blir det i utgangspunktet lagt mer vekt på å utvikle kommunikasjonsverktøy enn å utvikle muligheter for å sette milepæler.

3 Design

3.1 Introduksjon

Dette kapitlet inneholder informasjon om design på systemet vi har utviklet, både grov oversikt over design og detaljert informasjon om komponenter og grensesnitt.

3.1.1 Generelle mål

Målet med designet var å utvikle et oversiktlig miljø som er lett å bruke, samt at systemet skulle være så effektivt som mulig. Vi har tatt utgangspunkt i et eksisterende førstepersons skytespill (GTest), siden dette har en design som gir funksjonalitet og effektivitet, samtidig med brukervennlighet.

3.1.2 Kort beskrivelse av systemet

Vi bestemte oss for å bruke grafikkmotoren Genesis3D og spillet GTest for å utvikle prototypen vår. Vi har tilpasset og modifisert GTest slik at det møter våre krav og vi har utviklet et scenario ved hjelp av verktøy som er tilpasset denne motoren, som vi kjører i prototypen. Vi har implementert en meny som man bruker før man har startet "spillet" og en "in-game" meny. Prototypen har to forskjellige roller man kan velge blant. Disse er kontrollør og skadestedsleder. Scenarioet har et hendelsesforløp som blir styrt av kontrolløren. Systemet kan kjøres som en klient eller server. I servermodus kan systemet koble til eksterne klienter via nettverk la disse "spille" via et multiplayer-system.

3.1.3 Bruksområde

Dette systemet skal brukes av Sivilforsvaret som en demonstrasjon på hva som er mulig å gjøre med den teknologien som finnes i dag. Systemet kan brukes som grunnlag for videreutvikling eller som et eksempel på en løsningmetode. Dette systemet skal ikke distribueres på noen som helst måte.

3.1.4 Rammer

På grunn av tidsrammen vil vi begrense detaljene i grafikken og konsentrere oss mer om effekter, kunstige aktører og hendelser i scenariet. Systemet skal kjøres på maskiner med eldre hardware, dvs. maskiner på 400-800 Mhz med liten støtte for tung grafikk. Av den grunn må systemet være effektivt.

3.2 Valg av grafikkmotor

3.2.1 Hva er en grafikkmotor?

En grafikkmotor er et verktøy som brukes av applikasjoner, f.eks. spill, til å generere grafikk på skjermen. En motor inneholder som regel en rendrer, et partikkelsystem, kollisjons-deteksjon, et vektorsystem for posisjonering og andre matematiske oppgaver, system for å håndtere fysikk, system for håndtering av lyd, filsystem, system for feilrapportering og egen datastruktur.
Rendreren er den delen som tegner opp ting på skjermen fortløpende, bilde for bilde.
Partikkelsystemet brukes til å generere alle partikler som brukes i grafikken, f.eks. røyk, flammer, støv, regnvær, noen former for lys, tåke osv.
Kollisjons-deteksjon brukes for å registrere om objekter, f.eks. i et spill, kolliderer med hverandre. Dette systemet registrerer bare en kollisjon, du må selv bestemme hva som skal skje når noe kolliderer.
Vektorsystemet brukes for å posisjonere objekter og elementer i det rommet eller flaten som skal rendres. Brukes også til mange andre matematiske oppgaver.
Fysikk-systemet brukes for å generere realistiske omgivelser. Fysikk-systemet inneholder f.eks. gravitasjon. Mange spill o.l. bruker ofte en egen fysikkmotor for å generere disse tingene, men som regel finnes det et system for dette i grafikkmotorer.
Lydsystemet brukes for å spille av lyd-filer.
Feilrapporterings-systemet brukes under utvikling av en applikasjon som bruker grafikkmotoren. Det er et verktøy som brukes for å rapportere feil mens man tester applikasjonen. Ofte logges feil til en logg-fil.
En egen datastruktur er datastruktur er ofte en fordel. Denne forenkler ofte prosessen med å utvikle en applikasjon som bruker grafikkmotoren siden datastrukturen er tilpasset den type utvikling. Man har ofte egne typer variable slik som objekter, vektorer, formater og indekser.
En grafikkmotor er også ofte tilrettelagt for nettverkskommunikasjon.
Alt dette er organisert som et bibliotek av funksjoner, data-strukturer og klasser som man kan bruke i en applikasjon, f.eks. et spill.

3.2.2 De forskjellige grafikkmotorene vi vurderte

Vi vurderte mange motorer til dette prosjektet. Disse var: Genesis3D, Ogre3D, Crystal Space 3D og Reality Factory.

Grafikkmotor	Egne verktøy for utvikling av topologi og objekter	System for skripting	Open Source (gratis)	Krav til maskinvare	Ressurser/hjelp på internett	Ferdig utviklet og grundig testet
Genesis3D	Ja	Nei	Ja	Lave/Middels	Mye	Ja
Reality Factory	Ja	Ja	Ja	Middels	Mye	Ja
Crystal Space 3D	Nei	Nei	Ja	Høye	Lite	Nei
Ogre3D	Nei	Ja	Ja	Høye	Lite	Nei

Figur 3Vurdering av forskjellige motorer

Ogre3D: denne grafikkmotoren er ikke beregnet på utvikling av 1. persons spill, men er utviklet for brukes i andre spill og applikasjoner. Den fokuserer mer på å generere veldig detaljert grafikk og kompliserte former. Vi har valgt å fokusere mer på funksjonalitet og komplisert og detaljert grafikk er ikke noe mål for oss. Derfor valgte vi å bruke en grafikkmotor som er mer skreddersydd til vårt formål enn det Ogre3D er.
En annen ting vi så som en ulempe er at Ogre3D er en «ung» motor og det er få som har erfaring med å bruke den til 1. persons 3D spill.
En fordel er at den er gratis å bruke under LGPL lisensen. For mer informasjon: http://www.ogre3d.org .

Crystal Space 3D: er en kompleks grafikkmotor med støtte for det meste som finnes av funksjonalitet i forbindelse med grafikkmotorer. Den er også utviklet for å generere komplisert grafikk og kompliserte former, og er beregnet på å lage spill med realistisk utseende. Det er mange positive ting ved denne motoren. Her er noen av dem:
Det finnes et system for skripting av hendelser, motoren er plattform-uavhengig (den kan brukes på windows, Linux og mac), den er gratis under LGPL lisensen.
Denne motoren er under utvikling og er ikke helt ferdig. Den utvikles av en stor gruppe frivillige som jobber sammen via Internett. Vi så på dette som en ulempe siden det ikke finnes god dokumentasjon på hvordan man skal bruke motoren og den er ikke testet fullt ut ennå. Vi følte vi burde bruke en motor som var kvitt de fleste barnesykdommer og som ikke var så kompleks. Derfor valgte vi å ikke bruke Crystal Space 3D. For mer informasjon om Crystal Space 3D: http://crystal.sourceforge.net/tikiwiki/tiki-view_articles.php .

Genesis3D: dette er motoren vi valgte. Fordeler og ulemper med denne og begrunnelsen for dette valget er grundig beskrevet i neste avsnitt, *2.3 Hvorfor vi valgte Genesis3D.*

Reality Factory: Denne grafikkmotoren er en videreutvikling av Genesis3D og inneholder endel funksjonalitet som ikke Genesis3D har. Den har blant annet et system for skripting av hendelser og den er utvidet, i forhold til Genesis3D, slik at den kan generere mer detaljert grafikk. Den har også et mer avansert system for kollisjons-deteksjon. Grunnen til at vi ikke valgte å bruke denne er at den krever mer maskinkraft enn Genesis3d. Dessuten hadde vi problemer med å kompilere Reality Factory.
For mer informasjon om Reality Factory: http://www.realityfactory.ca/v3/ .

3.2.3 Hvorfor vi valgte Genesis3D

Genesis3d v1.1 var den av de motorene vi vurderte som hadde flest fordeler. Den er utviklet for å brukes til 1. persons spill og inneholdt all funksjonalitet vi trengte.
Fordelene med Genesis3D:

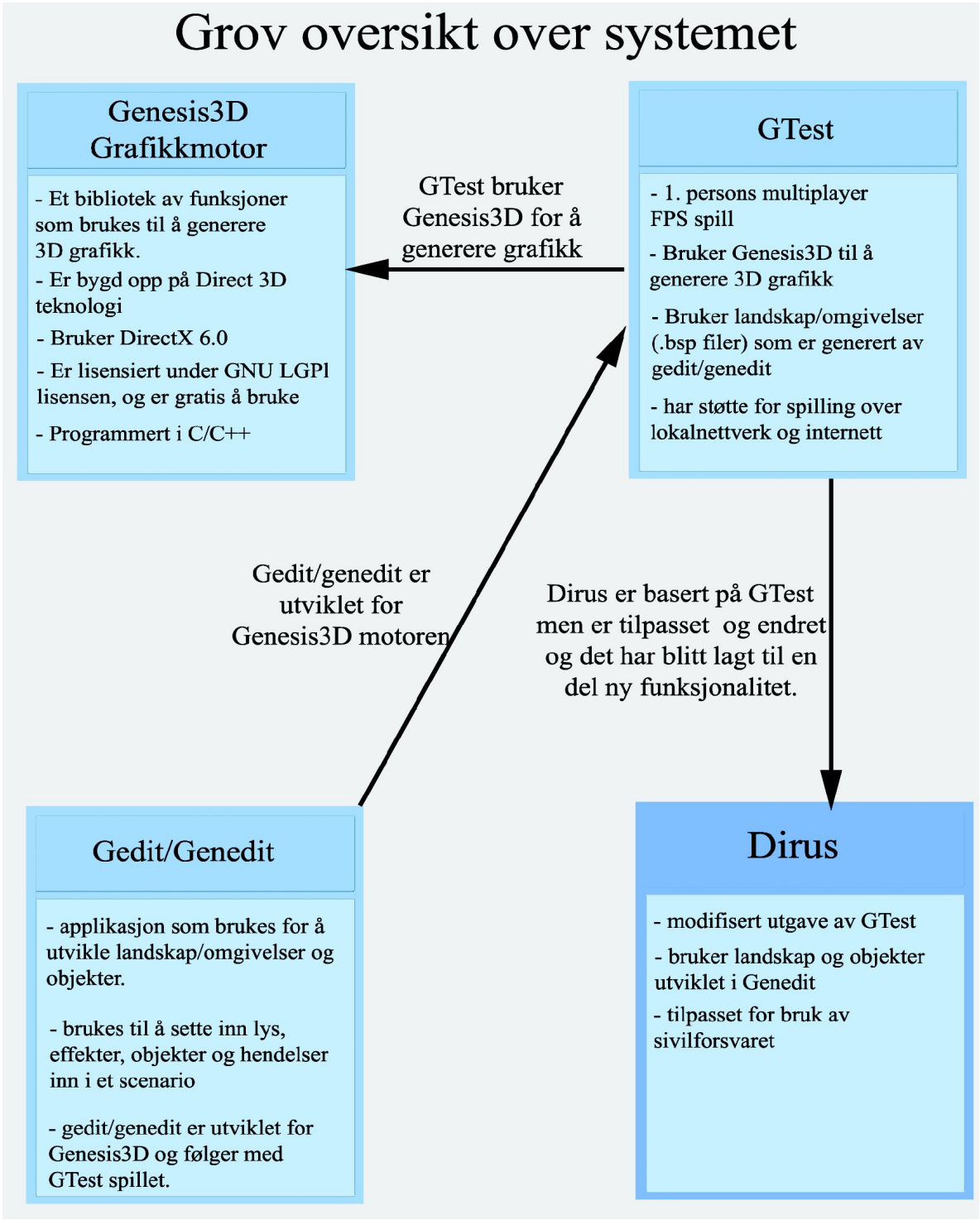
- Det er en relativt gammel motor, og krever derfor ikke for mye av hardware og software. Den er bygd på DirectX 6.0 SDK (software development kit), som også er kompatibel med eldre skjermkort og hardware. Dette er viktig siden sivilforsvaret ikke har en flunkende ny maskinpark som er beregnet på å kjøre 3D grafikk.
- Siden det er en eldre motor, så finnes det et stort miljø rundt denne motoren. Den har vært brukt relativt lenge, og mange har laget spill basert på motoren. Det er egne sider på nettet for tutorials, artikler og nyheter angående motoren, samt forum der andre som brukeren motoren legger inn sine spørsmål og svar. Dette har vært til stor hjelp, siden 3Dgrafikk-motorer er kompliserte og det tar tid å sette seg inn i dem.
- Denne motoren er også brukt i et undervisnings-prosjekt som tar for seg utvikling av spill. Dette prosjektet heter ProjectZ(http://www.genesis3d.com/~seven/projectz/projectz.html). Dette er også en viktig ressurs når man skal forstå hvordan en grafikkmotor fungerer. (ProjectZ er et privat prosjekt, og hører ikke til noen undervisnings-institusjon).
- Genesis3D er gratis under LGPL lisensen. Dette er viktig siden Sivilforsvaret ikke har noe budsjett for dette prosjektet. Denne lisensen gir oss også frie tøyler til å bruke kildekoden som vi vil, så lenge vi opplyser om hvor vi har fått den fra.
- Det finnes et åpen kildekode-spill som heter GTest som bruker Genesis3D-motoren. Dette er et multiplayer spill og det er innebygd støtte for å spille over LAN eller over Internett, via TCP/IP protokollen. GTest er ofte brukt i tutorials på nettet og mange bruker GTest som et utgangspunkt når de skal lage sitt eget spill. Vi har tatt utgangspunkt i GTest, siden dette sparte oss for mye arbeid, og da kunne vi konsentrere oss mer om funksjonalitet og hendelser i scenariet. Vi slapp blant annet å utvikle nettverkskommunikasjon, som er en komplisert jobb, siden dette allerede finnes i Gtest.
- Det er også utviklet en del verktøy som kan brukes i sammenheng med Genesis3D. Det finnes en WorldBuilder som brukes til å utvikle landskap og bygninger og en ActorBuilder/ActorStudio som brukes til å utvikle aktører/personer og objekter som puttes inn i et scenario. Genesis3d er også kompatibel med kjente programmer for å utvikle tre-dimensjonale figurer, slik som 3DstudioMax og Milkshape3D. Dette er en stor fordel.
- Ulemper: Det er for oss én stor ulempe med Genesis3d motoren. Den er utviklet for å fungere best i innendørs-scenarier. Det vil si at den ikke er så effektiv når man lager store utendørs-scenarier. Det finnes en del tiltak man kan gjøre for å overkomme dette problemet. Noen av disse er å bruke far-clipping sammen med distance-fog (fører til at detaljer på en forhåndsbestemt avstand ikke blir tegnet opp), man kan skalere ned alt i scenariet slik at motoren slipper å rendre mange store ting og man kan gjøre om alle objekter til actorer, siden disse er mye raskere å rendre.

Med det utgangspunktet vi hadde og den tiden vi har til å utvikle Dirus, vurderte vi det slik at det var mange flere fordeler enn ulemper med Genesis3D-motoren, og det var ingen av de andre motorene vi vurderte som hadde like mange fordeler som Genesis3D. Det var viktigere for oss å levere Sivilforsvaret noe som fungerer, istedet for å komme med noe effektivt når det gjelder ytelse, men halvferdig. Denne prototypen (Dirus) skal ikke brukes til opplæring, men det skal være en demonstrasjon på hva som er mulig ved bruk av denne teknologien. Av denne grunn følte vi at Genesis3D var det beste alternativet for oss.

3.3 Overordnet design og system arkitektur

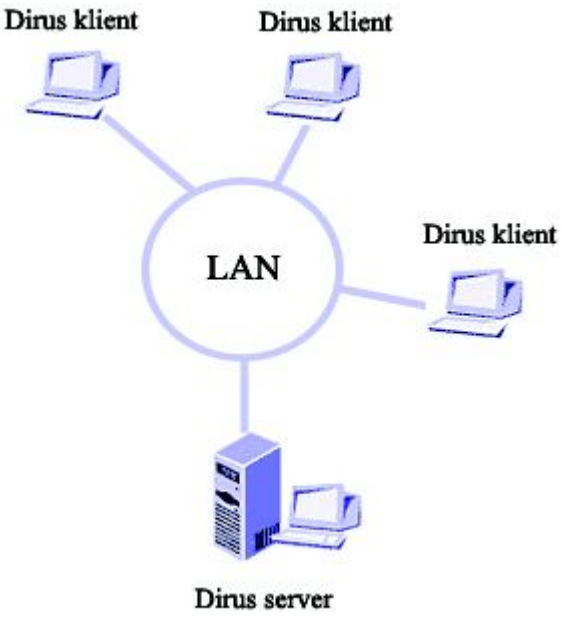
3.3.1 Grov beskrivelse av systemet og arkitekturen

Prototypen vi har utviklet i løpet av dette prosjektet er basert på et førstepersons skytespill som heter GTest. GTest bruker Genesis3D grafikkmotoren for å generere grafikken og er bygd opp rundt denne. Vi har utvidet og forandret på kildekoden til GTest for tilpasse den til vårt formål.



Figur 4 Grov oversikt over systemet

Dirus følger klient/server-arkitekturen hvor man, etter å ha startet Dirus, kan enten være klient eller server. I server modus så oppretter man en server på sin maskin og man får rollen som kontrollør. I klient modus så logger man seg på en server som er opprettet av en kontrollør og man får rollen som skadestedsleder. Dette fungerer slik at serveren holder rede på informasjon fra spillets gang, hendelser, aktørenes posisjon og lignende. Denne informasjonen distribueres så igjen til alle klientene. Kommunikasjon over nettverk foregår via TCP/IP-protokollen. Dette er den mest brukte protokollen, noe som innebærer at man kan kjøre Dirus i nær sagt alle nettverk.



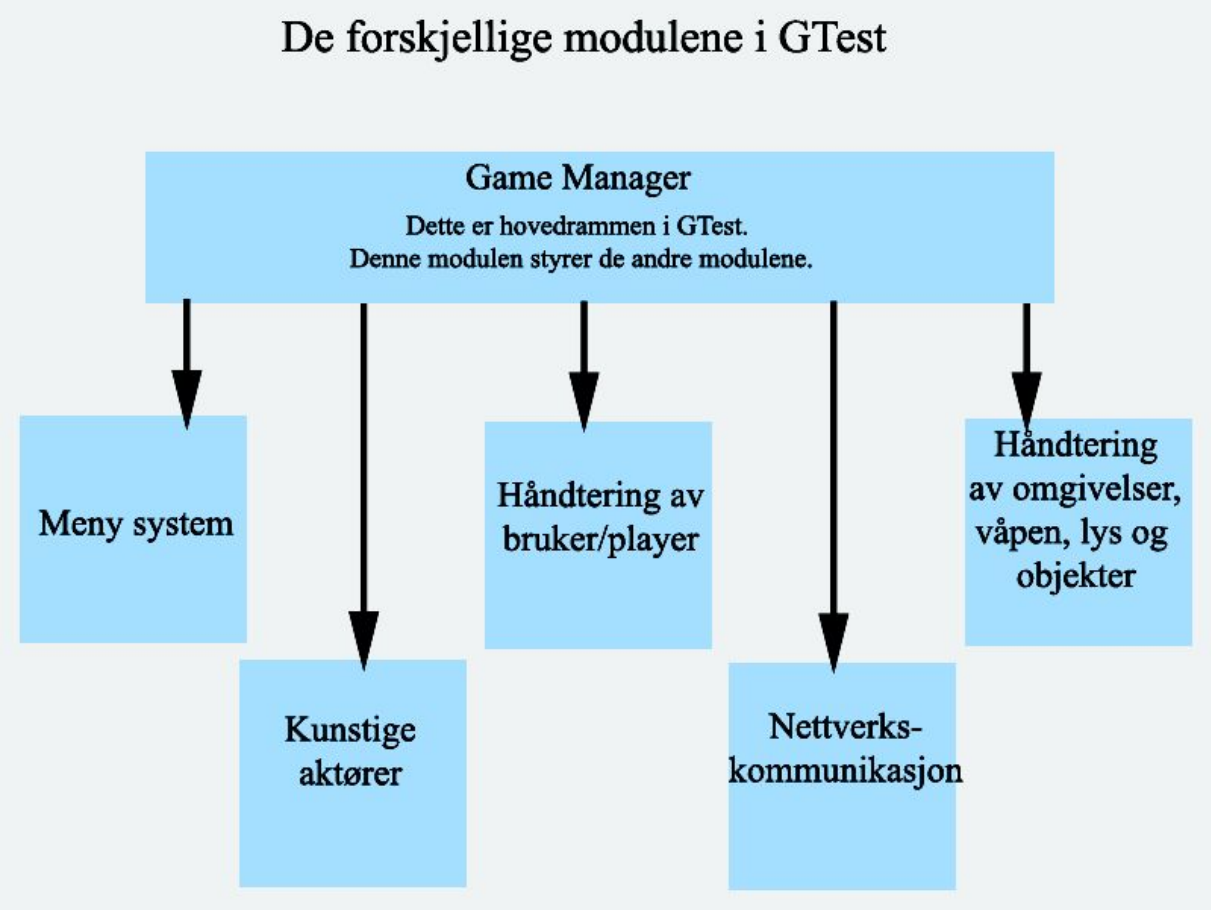
Figur 5 Nettverkstopologi for systemet

3.3.2 Grundig beskrivelse av systemet

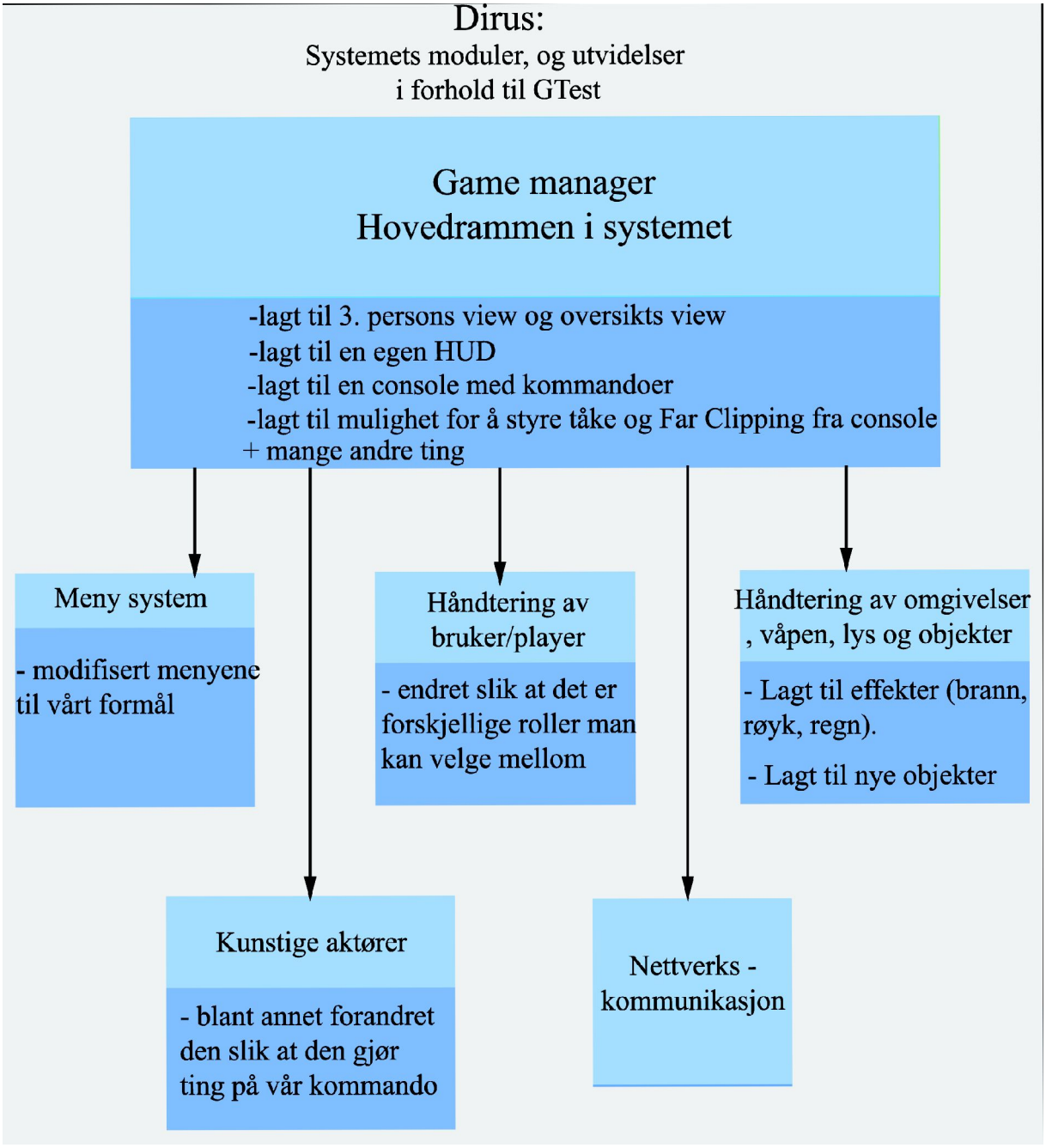
- **Genesis3D:** er en Open Source grafikkmotor som blir brukt til å utvikle 3D-spill med. En grafikkmotor er et bibliotek med funksjoner, datastruktur og objekter som brukes til å generere grafikk.
  Genesis3D rendrer real-time, og har alle funksjoner man har bruk for for å utvikle et moderne 3D-spill. Hovedgrunnene til at vi valgte å benytte denne motoren fremfor andre alternativer, var at Genesis3D er en mye brukt motor, det finnes derfor mye dokumentasjon og brukererfaringer på Internett til denne. Dette er en Direct3D-motor og er utviklet med DirectX 6.0 SDK (Software Development Kit) som er et utviklingsverktøy for Direct3D applikasjoner. DirectX 6.0 SDK er utviklet av Microsoft.

  For grundigere info om Genesis3D: www.genesis3d.com.

- **Gtest:** er et førstepersons skytespill som bruker Genesis3D som grafikkmotor. Dette spillet laget for å vise hva motoren er god for og brukes av mange for å lære seg å utvikle et 3D-spill. Av den grunn finnes det mye ressurser på internett som beskriver hvordan man skal forbedre og forandre på dette spillet. Dette er en av grunnene til at vi valgte denne motoren og GTest som utgangspunkt.
  GTest er i utgangspunktet et multiplayer spill. Det vil si at flere personer logger seg på en server og spiller mot hverandre, enten i et lokalnettverk eller på internett. GTest kommuniserer via TCP/IP-protokollen. Det er mulighet for å ha bevegelige objekter i omgivelsene man spiller i, som f.eks. dører og heiser.
  Det finnes også mange forskjellige objekter med forskjellige egenskaper som gjør opplevelsen mer dynamisk, slik som forskjellig lys, våpen, førstehjelpsskrin og rustning.
  Det finnes også såkalt intelligente kunstige aktører i GTest. Denne intelligensen finnes i de kunstige motspillerene som man kan spille mot hvis man ikke har noen å spille med eller ikke har forbindelse med noe nettverk. Disse er designet for å oppføre seg som en virkelig motspiller som gjør alt de kan for å ta livet av deg. Det er ikke noen avansert form for intelligente motspillere, men de gjør at man kan ha motspillere uten med hjelpere. GTest har også et eget menysystem.
  Grafisk fremstilling av hvordan GTest er bygd opp:



Figur 6 Grafisk oversikt over modulene i GTest

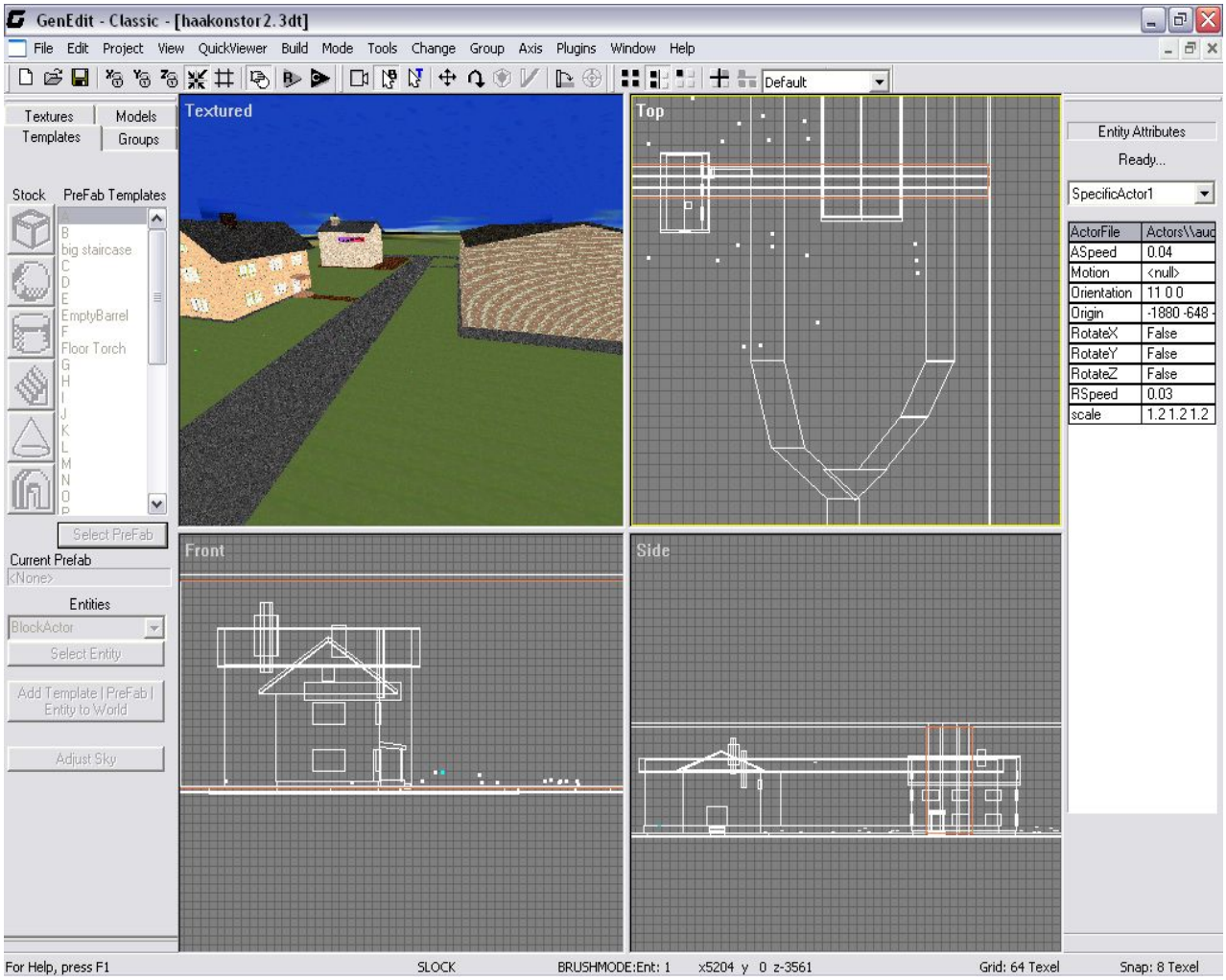


Figur 7Oversikt over modulene i Dirus

Figur 6 viser en grafisk oversikt over modulene i GTest og Figur 7 viser en grafisk oversikt over de samme modulene i Dirus. Hvis disse sammenlignes så får man et inntrykk av hvor i systemet vi har implementert de forskjellige tingene.

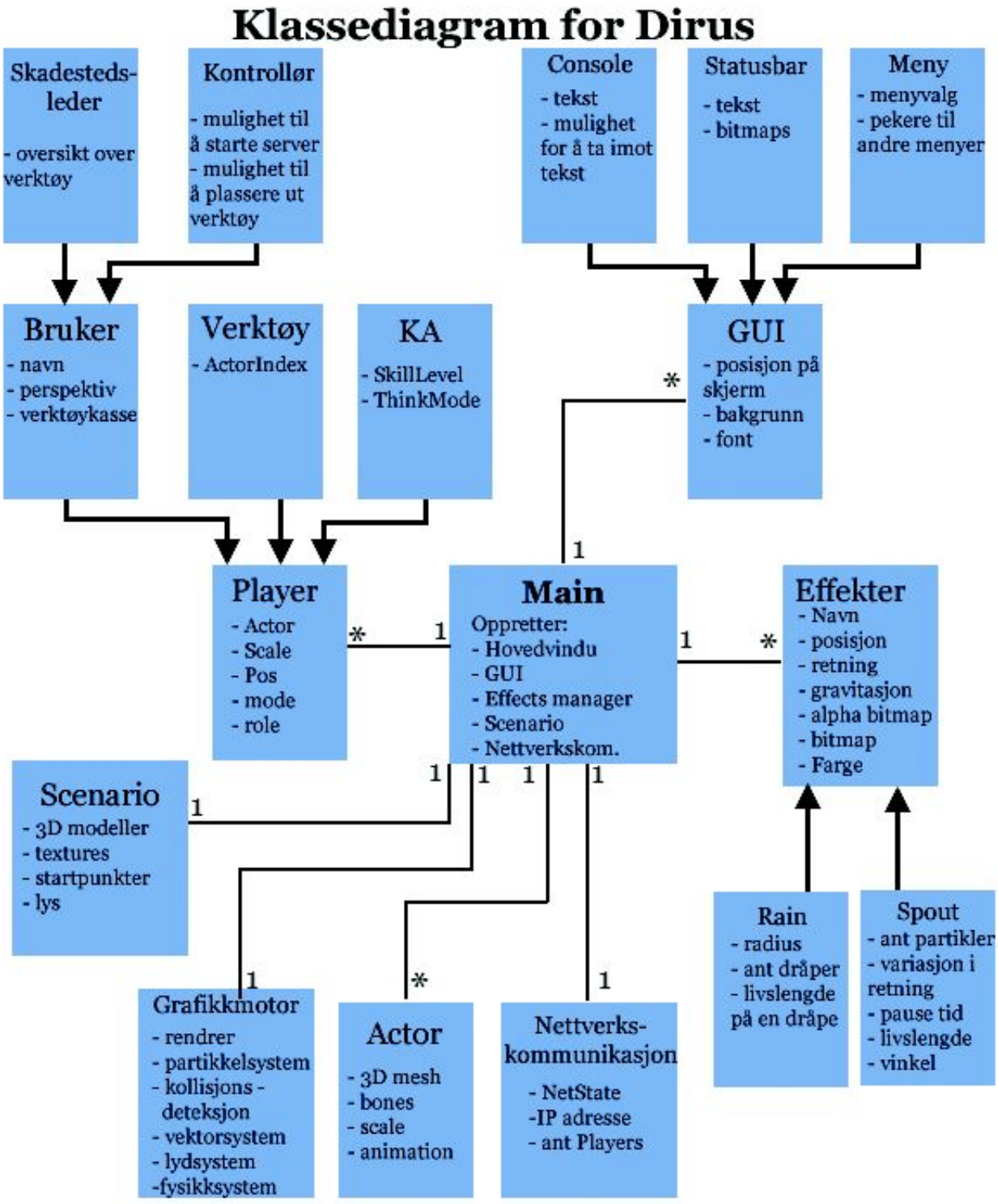
- **Dirus:** Vår prototype som er en modifikasjon av GTest. Vi har beholdt det meste av funksjonaliteten som fantes i GTest fra før og vi har lagt til en del for å tilpasse den best mulig til vårt formål. Vi har brukte mye tid på å studere kildekoden til GTest for å kartlegge hvordan systemet er bygd opp og hvordan datastrukturen er. Dette er de store modifikasjonene vi gjorde og de elementene vi la til GTest for å utvikle Dirus:
  - Vi har modifisert menysystemet slik at det inneholder de valg vi trenger, samt at utseende er forandret slik at det får en egen identitet.
  - Vi har endret systemet slik at man kan ha to forskjellige roller i et scenario, skadestedsleder og kontrollør. Disse rollene har forskjellige muligheter og rettigheter i et scenario slik at bruken av dette virtuelle spillbordet blir lik bruken av det virkelige modellbordet som Sivilforsvaret bruker. Kontrolløren er usynlig i scenariet slik at ting blir så realistisk som mulig. Det er også mulig å utvide med flere roller.
  - Det er gjort store endringer i oppførselen til aktørene, slik at vi kan få de til å ha forskjellige roller i et scenario. Noen er fagpersonell og andre er publikum, og de kan gjøre ting på vår kommando, som f. eks. at de kunstige aktørene som er publikum trekker automatisk mot samleplass for evakuerte når dette flagget blir plassert ut i scenariet.
  - Det er også mulighet for å skifte mellom tre perspektiver når man spiller, og disse er 1. persons- , tredjepersons- og overblikks-perspektiv. I overblikks perspektivet så har man oversikt over hele scenariet.
  - Dirus har støtte for flere effekter slik som branner, røyk, regnvær og det er støtte for nye objekter. Disse fantes ikke i GTest, så vi måtte legge til disse effektene selv. Disse blir plassert ut i scenariet i Genedit.
  - For å bedre ytelsen så har vi benyttet oss av distance-fog og far-clipping. Distance-fog begrenser synsfeltet til aktøren, og far-clipping innebærer at det aktøren ikke ser ikke blir rendret. På denne måten vil det kreve mindre maskinkraft for å spille. Modellene vi har laget er dessuten laget så enkle som mulig for å bedre ytelsen i spillet.
  - Det er også lagt til en Console. Dette er en tekstbasert kommandolinje som kommer ned på skjermen når man trykker på en spesiell tast. På denne kommandolinja så kan man skrive inn kommandoer som blant annet styrer distance-fog og far-clipping.
    Her er det fult mulig å utvide med nye kommandoer.
  - Det er også lagt til en helt ny statusbar som gir spilleren informasjon. Denne statusbaren ligger nederst i skjermbildet og inneholder forskjellig informasjon. På statusbaren kan brukeren se hvilke taster som man kan bruke i scenariet, f.eks. for å pause scenariet. Statusbaren opplyser også om hvilket verktøy eller flagg du valgt, har mulighet til å velge og antallet verktøy du har igjen å bruke. Det er også mulig å vise meldinger, i tekstform, fra systemet til brukeren på denne statusbaren.
  - Nettverkskommunikasjonen fungerer på samme måte som i GTest, bortsett fra at man kan ha to forskjellige roller når man spiller. Når man starter en server så er man kontrollør og skal man logge seg på en server så blir man skadestedsleder. Kontrolløren blir, som nevnt ovenfor, usynlig i scenariet.
  - Dirus har også en pause funksjon som ikke finnes i GTest. Man kan trykke på 'Y' tasten, så settes scenariet i pause modus, dvs. at alt fryses i scenariet. For å sette igang scenariet igjen så trykker man bare på 'U' tasten.

- **Gedit/Genedit:** Gedit er et program som brukes til å utvikle landskap og omgivelser til 3d spill som benytter seg av Genesis3D grafikkmotoren. Genedit er en nyere verson av Gedit som har endel nye funksjoner og er lettere å bruke. Gedit/Genedit er flettet sammen med GTest på den måten at de bruker samme header-filer. Dvs. at hvis man definerer og koder inn et nytt objekt, effekt eller tilstand inn i kildekoden, så vil Gedit støtte denne automatisk og man kan bruke det i et scenario uten problemer.
  For å lage omgivelser i GenEdit må man bygge disse opp med en rekke bokser. Man bruker forskjellige typer bokser for å lage vegger, tak, trapper osv. Man legger på textures på disse boksene slik at de får et passende utseende, for eksempel gress, panel, himmel osv.
  Når man er fornøyd med landskapet, (hus, veier, terreng o.l.) så kan man begynne å plassere inn objekter, effekter og modeller (biler, mennesker og alle ting som ikke kun har rette flater). Modeller og objekter kan utvikles med andre verktøy, slik som for eksempel 3D Studio Max og Milkshape 3D, og deretter settes inn i scenariet. Når alt er på plass slik man vil ha det, kompilerer man scenariet. Når man kompilerer, så tar man all informasjon som finnes i filen (.3dt fil) og organiserer det i en .bsp-fil. Så starter man Dirus og laster inn denne filen. .3dt filene er filer som GenEdit bruker for å holde all informasjon om et scenarie/brett. En .bsp fil inneholder alle data om statiske enheter i et scenario. Disse dataene er organisert i bsp filen som et binært tre, og det er denne filen som Dirus bruker for å kjøre et scenario.



Figur 8 Skjermbilde av Genedit

3.4 Klassediagram og beskrivelse



Figur 9Klassediagram over Dirus

Figur 9 viser en oversikt over de viktigste klassene som Dirus inneholder. Alle disse klassene er innebygd i Dirus, bortsett fra motoren som er et bibliotek men funksjoner som alle klassene i Dirus benytter seg av.

- **Main:** Dette er klassen som starter Dirus. Her lastes driveren inn og hovedvinduet og GUI opprettes. Via hovedmenyen så startes et scenario og de objekter som behøves for å kjøre dette. Man starter også nettverkskommunikasjon i denne klassen.
- **Nettverkskommunikasjon:** Denne klassen oppretter en server eller kobler Dirus til en server, og den tar seg av all nettverkskommunikasjon.
- **Actor:** En Actor er et figur format brukt i Genesis3D. Disse figurene er som oftest mennesker, kjøretøy, trær o.l. Hver Actor blir opprettet fra en binær fil av formatet .act. Dette filformatet inneholder figurens oppbygning, utseende og bevegelser. Disse figurene blir utviklet i eksterne modellerings programmer, som f.eks. 3Dstudio Max, og pakket sammen til en act fil.
  En Actor kan, i motsetning til en vanlig modell utviklet i Genedit, være animert og bevege forskjellige deler av «kroppen» i forhold til hverandre. En actor kan dessuten ha krumme overflater, i motsetning til modeller. En Actor behandles som ett enkelt objekt av Genesis3D, i motsetning til f.eks. et hus som består av flere objekter satt i sammen til en enhet. Dette påvirker ytelsen i positiv retning, siden actorer av denne grunn krever mindre maskinkraft for å rendres. Vi har derfor valgt å benytte actorer også til trær, flagg, sperrebukker og andre statiske objekter. Hver enkelt Actor har et unikt navn som defineres når den opprettes. I tillegg kan den skaleres, slik at vi ikke er avhengig av å bruke den samme størrelsen på alle actorer.
- **Player:** en player er et dynamisk objekt som inneholder en actor. Alle roller i Dirus, kunstig eller bruker, arver endel attributter fra denne klassen. Eksempler på roller: en kunstig aktør, et verktøy eller en bruker.
- **Verktøy:** Brukere av Dirus har en «verktøykasse» som inneholder endel objekter som man kan bruke i et scenario. Disse objektene er forskjellige flagg som kan settes ut, f.eks. samleplass for skadde, venteplass for ambulanser, sperrebukk osv. Disse objektene har forskjellige funksjoner i et scenario. Når flagget «samleplass for evakuerte» plasseres ut i scenariet så trekker automatisk alle tilskuere mot dette flagget og samler seg der. Det samme skjer når «samleplass for skadde» plasseres ut, bortsett fra at de skadde aktørene trekker mot dette flagget.
  Selve figurene, flaggene og sperrebukken, er actorer.
- **Bruker:** Er selve spilleren. En bruker kan enten være skadestedsleder eller kontrollør. Disse to rollene har noen rettigheter til felles og noen rettigheter er forskjellige. f.eks. så har begge mulighet for å velge mellom tre forskjellige perspektiver i et scenario, mens det bare er kontrolløren som har mulighet til å plassere ut flagg og sperrebukker. Skadestedslederen har bare en oversikt over hvilke verktøy som er tilgjengelig. Hver bruker har også en figur i scenariet. Kontrolløren har en figur som er usynlig og skadestedsleder har en menneskefigur med grønn vest. Begge disse figurene er actorer. Kontrolløren skal være usynlig i scenariet, men må ha en figur allikevel, ellers så får man ikke opprettet en bruker.
- **Kunstig Aktør/Bot:** En KA (kunstig aktør) er definert som et dynamisk objekt i Dirus og styres ikke av spilleren, men av den kunstige intelligensen i Genesis3D. En KA kan være en skadet person, tilskuer, sykebil, brannbil, politibil o.l. Hver enkel KA har en variabel kalt «SkillLevel». Denne settes til en verdi som AI-en bruker til å styre KA-enes adferd. Hver enkelt KA-s startposisjon settes under modelleringen av scenariet. Hver KA har en figur som er en actor.

- **Skadestedsleder:** en skadestedsleder er en bruker som har mulighet til å logge seg på en Dirus server. I et scenario har skadestedslederen mulighet til å se hvilke verktøy som er tilgjengelig, via statusbaren, men kan ikke plassere ut noen av disse. Det er kontrolløren som skal gjøre dette.
- **Kontrollør:** en kontrollør er en bruker som har mulighet til å starte en Dirus server. I et scenario har kontrolløren mulighet til å plassere ut verktøy. Disse verktøyene velger kontrolløren via statusbaren og plasserer dem ut med et museklikk.
- **Grafikkmotor:** En grafikkmotor er et verktøy som brukes av applikasjoner, f.eks. spill, til å generere grafikk på skjermen. Motoren inneholder funksjonalitet for håndtering av all grafikk, brukergrensesnitt, menysystem, AI, partikkelsystem, posisjonshåndtering, objekthåndtering, kollisjons-deteksjon osv. Dirus bruker funksjoner og elementer fra motoren for å generere og opprette alle disse tingene. (grafikkmotoren er beskrevet grundigere et i tidligere kapittel).
- **GUI:** Er hva brukeren ser på skjermen. Består av to typer. 3D-delen, og 2D-delen. 3D delen plasseres på mesteparten av skjermen og viser selve scenarioet og det som skjer der. 2D delen er statusbar, samt tekst informasjon som legges oppå 3D-delen. I tillegg er det menyene, som også er av type 2D.
- **Console:** er en kommandolinje hvor brukeren kan skrive inn kommandoer til systemet (Dirus) mens man er inne i et scenario. Når man trykker på en tast dukker det opp et felt i toppen av skjermbildet hvor man kan skrive inn en kommando.
  I denne prototypen kan man kun justere distance fog via console.
- **Statusbar:** er et felt nederst på skjermen som gir brukeren informasjon i et scenario. Brukeren kan se hvilket verktøy som er valgt, hvilke taster som har funksjonalitet i et scenario og det er et felt helt til høyre på skjermen hvor tekstmeldinger vises.
  Statusbaren ligger der statisk og oppdateres kun når brukeren velger et nytt verktøy eller en melding kommer opp i meldingsfeltet.
- **Meny:** meny er den klassen som oppretter menyene, tegner opp bakgrunnsbilde og håndterer navigasjon i menyene.
- **Scenario:** Dette er omgivelsene hvor ulykken skjer i Dirus, og inneholder forskjellige tredimensjonale modeller av bygninger, veier osv. Alle flater i brettet har teksturer, som gir flatene ønsket utseende, f.eks treverk, mur osv. Det har også 3D-modellene (hus o.l.).
  Når man har opprettet et landskap og topologi, så kan man plassere inn forskjellige ting inn i scenariet, som f.eks. trær, biler, start punkter og effekter.
- **Effekter:** Dette er effektene som brukes i et ulykkesscenario. Disse effektene er brann, røyk, skum og regnvær. Denne klassen benytter seg av partikkelsystemet i motoren for å generere disse effektene.
- **Rain:** denne klassen arver fra klassen Effekt og bruker partikkelsystemet for å generere regnvær over et område i et scenario.
- **Spout:** denne klassen arver fra klassen Effekt og bruker partikkelsystemet for å generere flammer, røyk eller bobler i en posisjon i et scenario.

3.5 Design av brukergrensesnitt (UI design)

Brukergrensesnittet i denne prototypen bygger på det som allerede finnes i Genesis3D motoren og GTest. Hovedmenyen, som kommer opp når man starter programmet, er tilgjengelig for alle brukere. Vi satte opp forskjellige undermenyer for skadestedsleder og en meny for kontrolløren. Det finnes også "in-game"-menyer, dvs. at de er tilgjengelige under spillets gang. I selve spillet har man i tillegg diverse informasjon i forgrunnen av skjermbildet, også kalt statusbar.

3.5.1 Detaljert beskrivelse av brukergrensesnittet

Som nevnt ovenfor består brukergrensesnittet av forskjellige menyer, samt noe 2D -grafikk i forgrunnen av skjermbildet når man spiller (statusbar.)
Hovedmenyen bygger på grafikken og menyene som allerede finnes i Gtest, vi har byttet bakgrunnen, skrifttypene og valgene i menyene. Hovedmenyen kommer opp midt på skjermen med en gang prototypen har startet, rett etter Genesis3D's "splash-screen." På denne menyen kan brukeren velge om han/hun skal være skadestedsleder eller kontrollør, om man vil endre innstillinger som f.eks. endre navn på spiller og forandre skjermoppløsning, man kan også avslutte Dirus fra denne menyen. Menyvalget Credits er et menyvalg som kreves i følge lisensen til Genesis3D. Dette menyvalget viser et bilde på skjermen hvor utviklerne av Genesis3D er listet opp. Alle disse valgene har undermenyer som har samme utseende som hovedmenyen, bortsett fra Credits menyvalget.

Hovedmeny: **Kontrollør**
Skadestedsleder
credits (må være med i henhold til lisensen)
Innstillinger
Avslutt

Hvis du velger å være kontrollør så kommer man inn i en meny hvor man kan sette opp en server. Serveren lytter på en spesifikk port etter tilkoblingsforespørsler fra klienter. Det er bare en kontrollør pr. scenario, og en eller flere skadestedsledere som logger seg på denne serveren. Denne menyen inneholder disse valgene:

Starte server meny: **IP adresse** (IP-adressen til server må ikke spesifiseres)
En liste med scenarier

Hvis man velger å være klient så kommer man inn i en meny hvor man kan koble seg til en allerede eksisterende server (som en kontrollør har startet). Her kan du skrive inn IP-adressen til serveren hvor scenariet kjøres og koble seg til den. Hvis scenariet kjøres på en server i samme lokalnettverk som din datamaskin, trenger man ikke å spesifisere IP-adresse. Det kjøres ingen autentisering på hvem som kan koble seg til en Dirus-server, så alle med Dirus og kart-filen som serveren kjører, kan koble seg til. Det kan forekomme problemer med tilkobling til serveren hvis denne befinner seg bak en brannmur.

Koble til: **IP adresse**
Connect

I tillegg til disse menyene er det en statusbar (2D-grafikk i forgrunnen av skjermbildet som er der hele tiden, uavhengig av hva som skjer i bakgrunnen). Denne statusbaren viser en del informasjon som både kontrollør og skadestedsleder har nytte av i scenariet. Den viser informasjon om hvilke knapper på tastaturet som har spesielle funksjoner, verktøyet/valget som er gyldig vises med bilde på statusbaren, meldinger og tekst som skal vises på skjermen vises i statusbaren.

3.5.2 Skjermbilder av menyer

Figur 10 Hovedmenyen og statusbar fra GTest som vi jobbet ut fra.

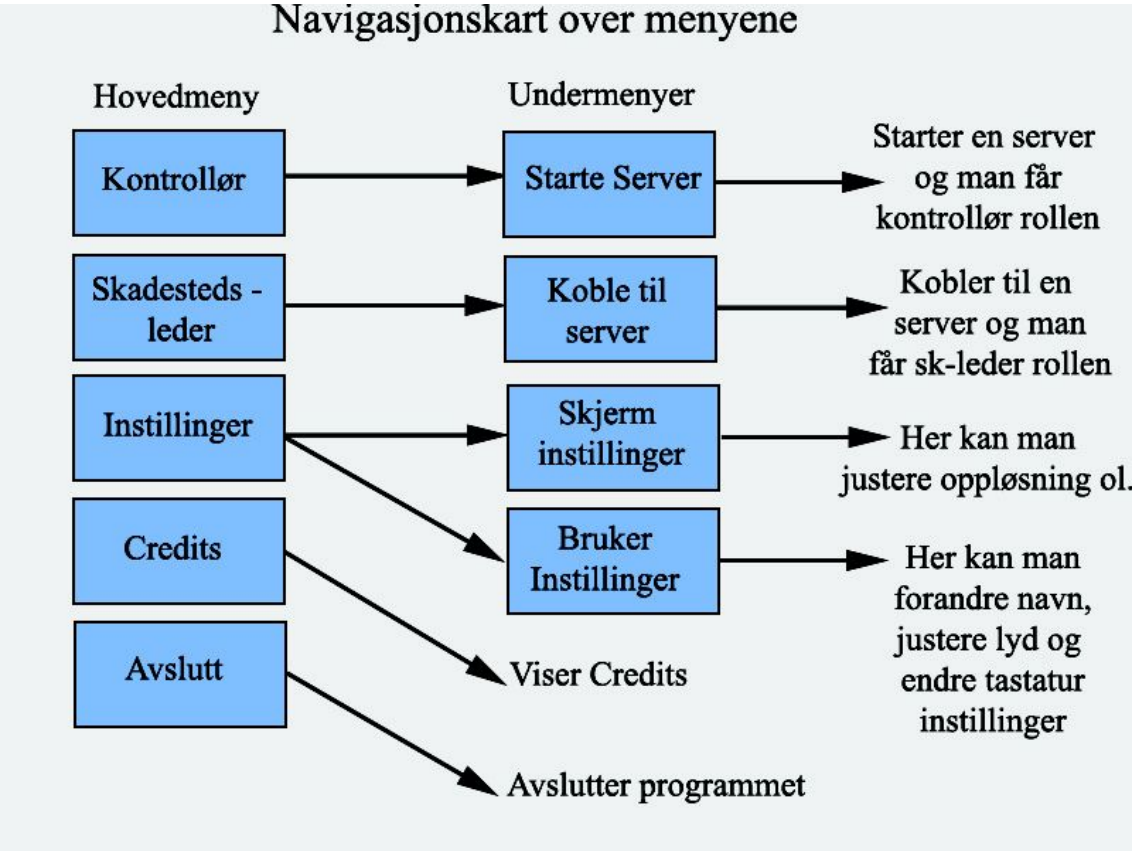
Figur 10viser menyen vi tok utgangspunkt i. Dette er menyen i GTest slik som den var originalt, uten bakgrunnsbilde.
Vi valgte å videreføre dette grensesnittet da det er brukervennlig og lettforståelig, dessuten enklere å for oss som utviklere enn hvis vi hadde laget et helt nytt brukergrensesnitt.
Man navigerer rundt i menyene via piltastene og trykker på enter-tasten for å velge et menyvalg. Et alternativ hadde vært å bruke data-musen for å navigere, men siden oppdragsgiver hadde presisert at detaljert grafikk og menysystemet ikke hadde prioritet så valgte vi å beholde den originale måten å navigere på. Vi forandret kun på bakgrunnsbildet, fonten på teksten i menyen og på menyvalgene. Fonten ble forandret slik at den passet bedre på den nye bakgrunnen.

Figur 11 Skjermbilde av Dirus menyen

Figur 11viser et skjermbilde av menyen slik den er i Dirus. Den har et bakgrunnsbilde som er lett og bytte ut, hvis man ønsker det. Denne menyen har andre menyvalg og annen font enn menyen i GTest.

3.5.3 Navigasjonskart over menyene

Figur 12viser et kart over hvordan man navigerer gjennom menyene i Dirus. Menysystemet har en hovedmeny og undermenyer. Hovedmenyen er den som kommer på skjermen når man starter Dirus. Navigasjonen er logisk og enkel, og den ligner på menyer som er vanlige i dataspill.



Figur 12 Navigasjonskart over hovedmenyen og undermenyer.

3.5.4 Beskivelse av statusbar

Figur 13 Beskrivelse av statusbar

Statusbaren er der for å gi brukeren informasjon under et scenario. Den består av både statiske og dynamiske elementer. Statusbaren er delt opp i forskjellige felter som har forskjellige funksjoner. Helt til venstre er det et statisk felt som viser hvilke taster på tastaturet som har funksjonalitet forbundet med scenariet, f.eks. hvilke taster som brukes for å skifte perspektiv, aktivere talekommunikasjon og åpne menyer.

Det midterste feltet viser hvilket verktøy/objekt som er valgt og kan settes ut via venstre museknapp. Her vises også antallet av disse objektene man har til disposisjon.
Feltet helt til høyre blir brukt til å vise informasjon som tekst. Her kan vi skrive ut all teskst som er relevant for brukeren. F.eks. “SAPL Evak-flagg satt ut.”
Statusbar bakgrunnen er tre bitmaper, en for hver del (delt ved de hvite strekene). Teksten på venstre delen skrives ut vet hjelp av det innebygde tekst-systemet far GTest. Det midterste feltet består av 2 elementer, et bitmap som representerer det nåværende flagget/sperrebukken, og et tall som skrives ut ved det innebygde tekst-systemet fra GTest. Feltet til høyre bruker tekst-systemet til å skrive ut status informasjon om hva som skjer i spillet. (se Figur 13)

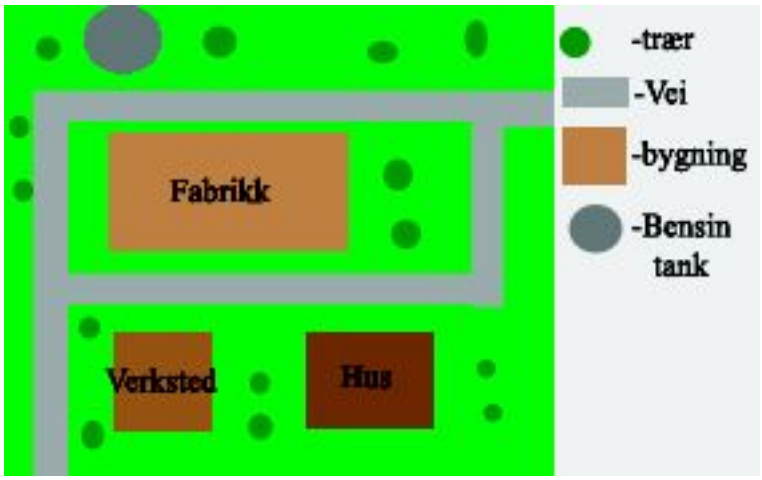
3.6 Design av scenario

3.6.1 Design av landskapet og bygninger

Vi har blitt enige med sivilforsvaret om å utvikle et enkelt scenario som er basert på et av de som blir brukt på det fysiske modellbordet på Starum. Den første gangen vi var på besøk på Starum så kom vi frem til at scenariet skulle inneholde en brann i en fabrikk i et tettsted. Vi bestemte oss da for å lage en kopi av landskapet de bruker på det fysiske modellbordet. Dette landskapet består av forskjellige bygg:

- en fabrikk
- et bolighus
- et lakkeringsverksted
- en stor tank med bensin(eller annen brannfarlig væske)

Brannen er i taket på fabrikken. Bolighuset har ingen funksjon, bortsett fra å begrense utsikt og plass i scenariet. Lakkeringsverkstedet, som inneholder brannfarlig lakk, har funksjonen at skadestedsleder skal vurdere om man skal iverksette noen tiltak for å sikre denne lakken mot brannen i fabrikken. Bensintanken har samme funksjon. Den er merket med en plakat som viser hva tanken inneholder, og skadestedsleder skal ta en avgjørelse ut fra dette. Ellers er det kjøretøy, trær og busker rundt omkring i scenariet slik at sikten besgrenses og at det ser ut som et vanlig boligstrøk. Vi har desuten satt inn ei kompassnål i himmelen til brettet, slik at aktørene kan bestemme vindretningen i forhold til himmelretningene.



Figur 14 Kart over scenariet

3.6.2 Hvilke dynamiske elementer scenariet inneholder

Scenariet inneholder noen dynamiske elementer. Disse er:

- Brann og røyk.
- Vindretning
- Fagpersonell (brannmenn, politi, medisinsk personell, skadde personer og publikum)
- Kjøretøy (brannbiler, politibiler, sykebiler og privatbiler)

Brann og røyk er plassert på fabrikkbygningen. Dette er altså brannen spillerne skal øve opp mot. Røyken som brannen danner, er av slik karakter at aktørene kan lese vindretningen ut i fra den.
Brannen vil ikke utvikle seg eller endre attributt på noen måte i spillets gang, noe som også gjelder vindretningen. Adferd til kjøretøy og fagpersonell blir styrt av Dirus, og kan påvirkes av kontrolløren ved utplassering av flagg. Ved utplassering av flagg, for eksempel SAPL Skadde (Samleplass skadde), ved utplassering av dette flagget vil alle actorer som er satt til "skadd" modus trekke mot dette flagget, og samles der.

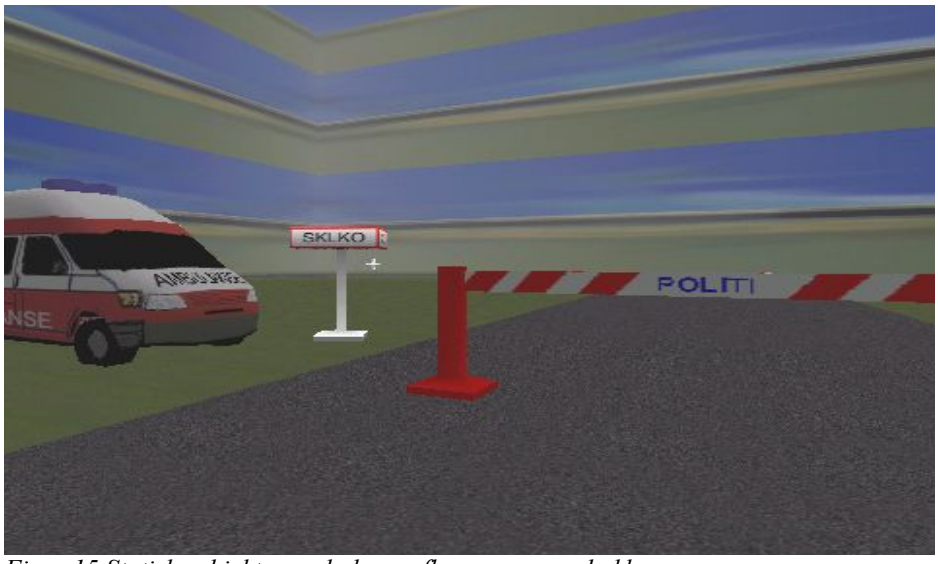
3.6.3 Hvordan scenariet ble utviklet.

Når vi skulle utvikle et nytt scenario så utviklet vi først et 3D-landskap. Dette konstruerte vi i Gedit eller Genedit. Siden Genesis3D er laget for innendørs scenarier, men vi i vårt tilfelle skulle lage et utendørs sceniarie, måtte vi bygge opp dette scenariet ved bruk av 2 veldig store rom satt inntil hverandre. Dette gjorde vi ved at vi i GenEdit lager først to hule bokser som vi satte til max størrelse motoren klarer, dvs. 4000x4000x4000 enheter (piksler). Vi prøvde å sette flere bokser inntil hverandre også for å få et enda større scenarie, men da fikk vi feil under kompilering av scenariet. Etter at vi satte disse boksene inntil hverandre var det ikke noen forbindelse mellom dem, de var fortsatt å betrakte som 2 store rom inntil hverandre. Det vi da måtte gjøre var å lage en massiv blokk(firkant) mellom de to store rommene, og så bruke en såkalt "cut brush" for å lage hull mellom. En cut brush fungerer nesten på samme måte som en solid kloss, men man merker av for at det skal være en cut brush, og da fjerner den alt som er innenfor det området den dekker. Man kan med denne da kutte bort deler av en solid kloss. Det vi da satt igjen med er et stort "rom" på 4000x8000x4000 enheter. Grunnen til at vi hadde det så høyt oppunder taket som 4000 enheter er fordi at da ble "overhead" view'en mye bedre og mer oversiktlig.
Når vi hadde gjort dette måtte vi lage textures til gulvet/bakken, vegger/horisont og tak/himmel. Dette gjorde vi i Adobe Photoshop, det vi da måtte passe på var at overgangen mellom vegger/gulv/tak ble minst mulig ettersom det skulle se ut som et utendørs scenarie. Bufferen som håndterer textures i Genesis3D 1.1 motoren har en begrensning slik at textures være ikke kan være over 8bit, dvs 256x256 piksler. Når vi da hadde laget ferdig disse texturene måtte vi legge de til i GenEdit's texture fil, denne finnes under /levels/ og heter gedit.txl. Denne filen inneholder alle textures som er klargjort og pakket for GenEdit. For å legge til texures i denne filen måtte vi bruke programmet tpack.exe. Når vi hadde gjort dette startet vi GenEdit på nytt, for å laste inn den nye filen og valgte hver enkelt flate/face og assignet hver enkelt texture til sin tilhørende flate. Så måtte vi strekke og skalere texturene sånn at de ble sittende riktig på vegger/gulv/tak.
Når vi hadde gjort dette, satte vi inn hus, trapper og veier ved å sette sammen firkanter trekanter og lignende man finner i GenEdit. Disse må også få textures på samme måte som vegger/gulv/tak.

3.7 Design av effekter, modeller og objekter

3.7.1 Statiske objekter

De statiske objektene har ingen animasjoner og var dermed lettere å lage enn dynamiske objekter. De statiske objektene er for eksempel biler, flagg og trær. Disse laget vi ved å modellere dem i *3D Studio Max* (andre programmer kan brukes). Så la vi på en tegning eller bilde utenpå den 3-dimensjonale modellen. Det kreves at man har ett "*bone*" for å få exportert objekter fra 3D Studio Max, dermed måtte vi lage ett av det. Etter dette ble gjort brukte vi et *MaxScript* for å lage en .nfo fil, en key fil. Nfo filen inneholder informasjon om 3D-modellen, slik som informasjon om mesh og bones og hvilke textures som skal være på figuren, mens .key filen inneholder animasjonen til figuren. Siste skritt er å bruke et program som heter "*Actor Creator,*" som klargjør modellene til bruk i Dirus. Actor Creator tar disse filene og bygger dem sammen til en actor.



Figur 15 Statiske objekter:ambulanse, flagg og sperrebukk.

3.7.2 Dynamiske objekter

De dynamiske objektene skiller seg fra de statiske ved at de også har animasjoner, dvs. at de er bevegelige(eks: At en person går). Disse objektene laget vi på samme måte som de statiske objektene, men i tillegg trengte de flere enn ett "bone", på en person måtte vi lage et skjelett av "bones" inni den 3-dimensjonale modellen for at vi skulle kunne lage animasjoner som gåing osv. Vi måtte så "binde" hver del av den 3-dimensjonale modellen til skjelettet for at den skulle bevege seg som vi ville. Det vi så gjorde var å lage en såkalt "walk-cycle" ved å flytte på skjelettet og merke av posisjonene på en tidsskala. Dette ble gjort på en slik måte at når man spiller av animasjonen fortløpende så den starter på nytt igjen etter den er ferdig, ser det ut som en kontinuerlig bevegelse. Når dette er gjort måtte vi eksportere på samme måte som for statiske modeller, men i tillegg måtte vi eksportere en .key fil som inneholdt informasjon om hvordan modellen skulle bevege seg. Vi måtte eksportere en .key fil for hver enkelt bevegelse modellen skulle ha. For eksempel en .key fil for gåing, og en .key fil for løping. Neste skritt i prosessen var å kompilere til en .act fil ved hjelp av "Actor Creator" igjen, men med dynamiske objekter måtte vi i tillegg til .nfo filen legge til hver enkelt .key fil før kompilering for at animasjonene skulle bli lagt til den 3-dimensjonale modellen.

3.7.3 De forskjellige modellene

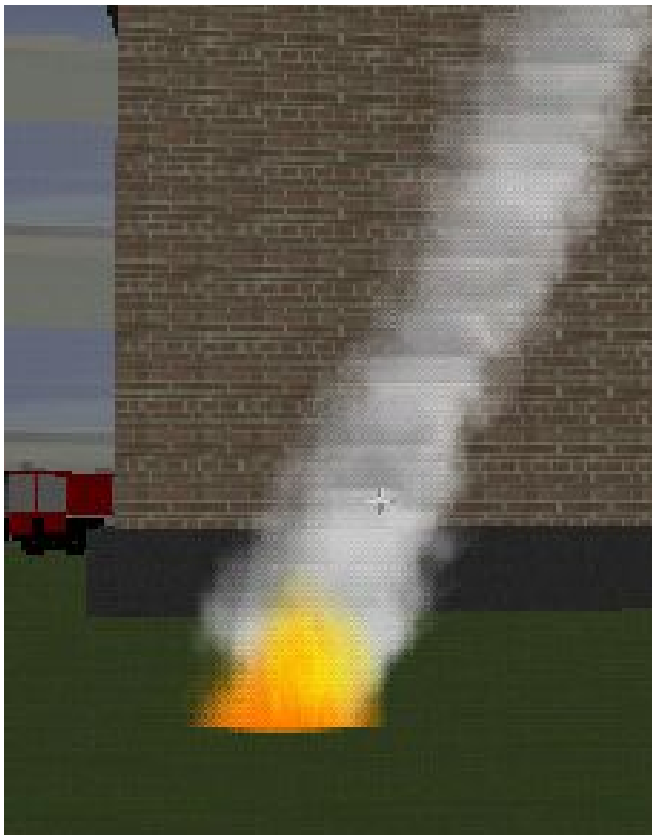
- Brannbil: Denne fant vi på internett, vi måtte forandre på litt for at den skulle fungere i prototypen, men vi sparte likevel en del jobb i forhold til å lage den helt fra bunnen selv. Denne kommer til å stå i bakgrunnen på scenariet og kommer ikke til å være animert.
- Politibil: Dette var den første modellen vi lagde helt fra bunnen selv. Det tok lengre tid enn forventet. Spesielt å lage bildene/tegningene som skulle legges på modellene tok lang tid. Denne kommer også til å stå i bakgrunnen på scenariet uten å være animert.
- Sykebil: Laget på samme måte som politibilen. Samme egenskaper, men med forskjellig form og tekstur.
- Personbil: Laget på samme måte som politibilen. Samme egenskaper, men med forskjellig form og tekstur.
- Flagg: I hovedtrekk likt som bilene, men en stor forskjell er at vi laget en flate som svever et godt stykke over flaggene som ikke har noen bakside, dvs. de blir usynlige fra undersiden. Dermed kan man se hva som står på flaggene fra overhead view. Dette er da spesielt nyttig for kontrolløren. (Se figur16)

Figur 16 Flagg sett ovenfra

- Kompassnål: Laget på samme måte som de andre statiske objektene.
- Brannmann: Her fant vi en actor som hadde alle animasjonene fra før. Så forandret vi på store deler av denne. Dette gjorde vi for å spare tid. Vi gjorde overkroppen mindre, siden han hadde voldsom overkropp. Vi lagde også ny texture for at utseendet skulle bli som en brannmann.
- Politimann: Basert på brannmannen, men forandret litt i form, og laget ny textur.
- Medisinsk personell: Utviklet på samme måte som Politimann.
- Tilskuere/publikum: Dametilskueren ble basert på "Dema" actoren som fulgte med Gtest. Men tatt bort våpen og forandret texturen.

3.7.4 Effekter

For å utvikle et scenario som er tilpasset Sivilforsvaret så trengte vi endel effekter som ikke fantes i GTest. Disse effektene var regn, brann og røyk. Disse effektene måtte kodes inn i protorypen vår, og tok i bruk partikkelsystemet i Genesis motoren.
Her fikk vi mye hjelp fra miljøet rundt Genesis3D motoren. Det fantes en tutorial på WOG (http://www.genesis3d.com/~wog/) sine sider som steg for steg gikk igjennom hvordan man bygde opp et system for å håndtere effekter og hvordan man utviklet effekter selv. Ved hjelp av denne tutorialen så implementerte vi en EffectsManager i prototypen vår og implementere i tillegg tre nye effekter. Disse effektene var sprites, spouts og rain. Sprites er statiske bitmaper som man kan bruke til forskjellige ting (flagg o.l.). Rain brukes til å generere regnvær, snø og haggelvær. Spouts brukes til å generere branner, røyk, bobler, skum o.l. Disse effektene ble implementert på en slik måte at de ble tilgjengelige fra Gedit og Genedit, som gjorde det enkelt å putte disse effektene inn i et scenario. For mer detaljert informasjon anngående implementering, se appendiks: dokumentasjon EffectsManager.



Figur 17 Skjermbilde av en kombinasjon av effektene ild og røyk.

3.8 Funksjonalitet og forbedring av ytelse

3.8.1 Endring av perspektiv

For å få forbedret oversikt over scenariet, at vi lagt til to andre måter å observere spillet på, i i tillegg til standard 1. persons view (L på tastaturet).

Chasecam: (K på tasteturet) Dette gjør at kamera i stedet for å være plassert i øynene på spilleren, er flyttet i bakkant av spilleren. Spilleren ser da brettet som før, men ser også seg selv. I tillegg får en litt bedre oversikt over brettet. Figur 18 Viser Chasecam.

Overhead view: (J på tastaturet) Gjør at kamera heves så høyt brettet tillater. Dette gjør at det er lett å få oversikt over hele brettet samtidig. Siden flagg er implementert med en flate som svever over de faktiske flaggene, er det fortsatt mulig å lese disse.

Figur 18 Chasecam

3.8.2 Pause funksjon

Etter ønske fra oppdragsgiver så har vi implementert en pausefunksjon i Dirus.
Denne funksjonen fungerer slik:
Når man er inne i et scenario så trykker man på tasten 'Y'. Da tegnes det opp et pausebilde og grafikkmotoren stopper og rendre. Da «fryses» scenariet mens Dirus fortsatt kjører i bakgrunnen og opprettholder nettverkskommunikasjonen bl.a. Når man trykker på tasten 'U', så startes scenariet igjen. Ulempen med denne løsningen er at man bare pauser scenariet lokalt på maskinen. Når man spiller over nettverk og man har logget seg på en server, så vil bare din figur i scenariet pauses, mens alt annet fortsetter å bevege seg fordi serveren ikke er satt i pause-modus. Hvis Serveren settes i pause-modus, så vil alt stå stille bortsett fra figurene til de som har logget seg på serveren. For å rette opp dette så må det sendes en melding over nettverket til alle maskiner som er koblet på et scenario, inkludert serveren, slik at alle setter seg i pausemodus når en av maskinene settes i pause-modus.
Det fantes ingen pausefunksjon i GTest fra før av, så vi måtte lage denne selv. Dette løste vi ved å legge til en funksjon i Dirus som stopper og starter all rendring fra grafikkmotoren. Denne funksjonen blir styrt av en boolean variabel. Denne kan enten være «true» eler «false». Når den er «true» så pauses Dirus og når den blir satt til «false» så startes Dirus igjen. Denne boolean variabelen styres via tastene 'Y' og 'U'.
For mer informasjon om hvordan vi løste denne oppgaven, se i dokumentasjonen angående pausefunksjonen i appendiks.

3.8.3 Console funksjon

Console er en kommandoprompt man får frem i Dirus ved å trykke 'Ø'. Consolen i GTest var kun en slags prompt for å se informasjon som spillet la ut. Det at consolen skulle ta i mot tekst fra oss var noe vi måtte legge til. Consolen har en int som viser hvilken possisjon på linja den er på. Dette gjør det enkelt å legge bokstavene som skrives i consolen fortløpende inn i en char array.

Dette kan så testes mot en rad med forhåndsdefinerte ord. Dette setter diverst bolean variable til tru eller false.

Dette systemet brukes av oss er bl.a. brukt av oss for å dynamisk regulere avstanden til distance fog (SETFOG x – der x er distansen). I tillegg kan den slås av/på ved bruk av “FOGON” og “FOGOFF”.

3.8.4 Adferden til de Kunstige Aktørene

KA-enes adferd er resultat av en modifisering av den opprinnelige KA-adferden i GTest. Denne opprinnelige AI-en hadde en fleksibel oppbygning, men ble bare satt til å utføre enkle oppgaver. Disse oppgavene bestod i å se etter spillere, skyte hvis innenfor rekkevidde og å følge en sti hvis ingen spiller er i synsfeltet. Vår modifisering av KA-ens AI bestod både av ny kode og endring av eksisterende kode. Systemet for å få KA-en til å følge en forhåndsdefinert sti benytter "pathpoints." Type og plassering av disse defineres på forhånd når brettet lages. Disse punktene kan kobles sammen til en sti, som en KA kan settes til å følge. Disse punktene kan være av forskjellig type. Vi har valgt å kun benytte en av dem, som gjør at KA-en stanser opp på det aktuelle punktet. Vi la også til en funksjon i AI-en, hvor vi kan skille på adferden til en KA basert på hvilken type personell/rolle den har i spillet. KA-ene kan være av fem forskjellige typer: publikum, skadde, politi-, ambulanse- eller brannpersonell. Alle KA-ene er på konstant utkikk etter flagg beskrevet i 6.5, og deres adferd bestemmes av hvilken type flagg som er satt ut. Hvis f.eks. spilleren setter opp et SAPL Skadde-flagg, vil alle KA-er av typen skadde, bevege seg mot dette flagget og stoppe opp når det har nådd flagget. Vi har brukt følgende beslutningstabell:

Flagg satt ut	Hvem	Modus
ingen	alle	Standard
SAPL Skadde	Skadde	Gå mot SAPL Skadde-flagg
SAPL Evak	Publikum	Gå mot SAPL Evak-flagg

3.8.5 Systemet for utplassering av flagg og sperrebukker.

Systemet for utplassering av flagg og sperrebukker er en modifisering av det eksisterende våpensystemet i GTest. Det ble tatt utgangspunkt i å modifisere rakettvåpenet, fordi dette lignet mest på hvordan vi så for oss at flaggsystemet ble løst. Slik raketten opprinnelig fungerte, spawnet den en rakett-actor, så ventet den en spesifisert tid, denne actoren ble slettet fra spillet, og en ny ble satt inn litt i fremkant av den gamle raketten. Ved å slå av denne funksjonen, fikk vi det slik at den første raketten hang igjen i lufta til "evig tid". Spilleren har en 4x4 transformasjonsmatrise som viser hvilken retning den til en hver tid ser. For at raketten skal peke i riktig retning i forhold til den retningen, får den samme transformasjonsmatrisen som spilleren. Dette gjør at flagget alltid vil stå i oppreist stilling, uavhengig om spilleren ser opp eller ned når det blir satt ut. Når rakettvåpenet var modifisert til ønsket flagg, var det bare å duplikere koden hos de andre "våpnene". Eneste som måtte gjøres i tillegg var bytte ut actoren som ble brukt. GTest hadde i utgangspunkt støtte for 4 våpen, så 3 ekstra var nødvendig å legge til ettersom vi skulle ha 6 flagg + sperrebukk. Dette ble løst enkelt ved å øke rangen på et par for-løker.
Når systemet var ferdig, ble flagg/sperrebukk tilgjengelig på denne måten:

1. SKLKO (Skadestedsleder Kommandoplass)
2. SAPL Skadde (Samleplass Skadde)
3. SAPL Evak (Samleplass Evakuerte)
4. Vent AMB (Venteplass Ambulanse)
5. HEL (Helikopter Landingsplass)
6. AKP (Ambulanse Kontrollpunkt)
0. Sperrebukk.

Figur 19 Eksempel på flagg - SAPL Skadde

3.8.6 TextPrompt entiteten

Denne entiteten gjør at man kan få en tekst til å komme opp på skjermen når man beveger seg i nærheten av et objekt av typen modell i scenariet. Modeller er stort sett dører, heiser og andre bevegelige elementer. Når man har plassert f.eks. en dør ut i et scenario, så kan man "hekte" på en TextPromt på den døren og bestemme hvilken tekst som skal komme opp på skjermen. Denne TextPromt-en blir trigget når spilleren beveger seg nær modellen.
Denne Entiteten fungerer ikke som den skal. Problemet er at når en TextPromt hektes på en modell som har en animasjon, f.eks. en heis, så starter ikke denne animasjonen.
Vi hadde tenkt å bruke denne entiteten for å gi brukeren, dvs. skadestedsleder, informasjon mens han/hun beveget seg gjennom scenariet. F.eks. kunne vi brukt denne entiteten for å opplyse skadestedsleder om innholdet i ei tønne i scenariet. Men siden vi ikke bruker modeller i scenariet, på grunn av ytelsen, så har vi ikke prioritert å løse problemet nevnt ovenfor.
For å løse dette problemet så må noe påvirkning på modeller fjernes fra denne entiteten, slik at den ikke stopper animasjonen til modellen den blir hektet på.
Men rammeverket for denne entiteten er implementert i Dirus og det er bare dette problemet som må løses.
For mer informasjon om hvordan TextPromt ble implementert, se dokumentasjon om TextPromt i appendiks.

3.8.7 Forbedring av ytelse

Siden Dirus skal kjøres på maskiner av noe forskjellige maskinkraft, gjorde vi systemet minst mulig krevende. Det er stort sett rendring av mange polygoner som tar mest maskinkraft, så vi iverksatte noen tiltak for å begrense antall polygoner som måtte rendres i et scenario. Vi gjorde to konkrete ting for å gjøre dette, som beskrevet under.

3.8.7.1 SpecificActor

I tillegg til hus og bygninger så ville vi ha andre statiske objekter i scenariet vårt. Vi begynte med å sette inn objekter som var laget med gedit og genedit. Disse objektene var satt i sammen av flere mindre objekter, og de ble tunge å rendre siden det var flere polygoner og flater, ofte flater som ikke synes, som skulle rendres. Det viste seg at det ble veldig tungt å kjøre et utendørs-scenario med slike objekter i, selv for en kraftig pc. Vi undersøkte på Genesis forumet på internett hvordan dette kunne løses og fant frem til at det gikk an å bruke .act filer (Actors) til dette. Actors brukes vanligvis til figurer, og kan også inneholde animasjoner. Disse filene inneholder færre polygoner og blir sett på som ett objekt av grafikkmotoren, og krever mye mindre maskinkraft for å rendres. Dermed implementerte vi en entitet som heter SpecificActor. Når denne entiteten settes inn i et scenario så kan du bestemme hvilken actor fil som skal brukes, f.eks. et tre, en bil eller lignende, og denne kan plasseres hvor som helst i scenariet. Man kan sette inn så mange SpecificActors som man vil i et scenario. Det er en ulempe med SpecificActor: Det finnes ingen blokkering av spilleren i den, og av den grunn kan man gå rett igjennom det objektet som settes ut. Dette må løses ved å utvide kollisjonsdeteksjons algoritmen til brukeren slik at den blokkeres når den møter på en actor.

For mer informasjon om hvordan vi implementerte SpecificActor, se i dokumentasjonen i appendiks.

3.8.7.2 Distance Fog og Far Clipping

For å unngå å rendre detaljer på lang avstand, noe som krever en del maskinkraft, så tok vi i bruk Distance Fog og Far Clipping. Distance Fog er rett og slett ei tåke som begrenser synslengden og Far Clipping sørger for at det du ikke ser, det blir ikke rendret. Vi begynte med å implementere Distance Fog fra bunnen av. Når vi var ferdige med dette så fant vi ut at dette allerede fantes i motoren og det var løst litt bedre enn vi hadde fått til. Far Clipping fantes også i motoren, så da tok vi disse i bruk. Vi implementerte et system for å styre Distance Fog og Far Clipping i Dirus, og disse kan styres fra Console.

For mer informasjon om implementeringen av dette, se i dokumentasjonen i appendiks.

3.8.8 Kommunikasjon mellom brukerne

Til kommunikasjon mellom brukerne har vi valgt å bruke et program som heter Ventrilo. Med dette programmet kan man bruke et mikrofon og høytalere til å kommunisere mellom datamaskiner. Dette gjøres ved at man setter opp en server for Ventrilo på en maskin. Så bruker man klientprogrammer på flere maskiner til å koble opp mot denne serveren. Man kan så snakke sammen via mikrofon og høyttaler. Dette programmet starter man og kobler seg til en server før man starter Dirus. Deretter starter man Dirus og når man skal kommunisere med de andre så trykker man på en forhåndsbestemt tast og snakker i mikrofonen. Da hører alle som har logget seg på serveren hva du sier. Det går også an å snakke flere på en gang, siden kommunikasjonen går i full duplex. I utgangspunktet hadde vi planer om å implementere en chat-funksjon inn i statusbaren slik at brukerne av Dirus kunne skrive til hverandre. Men dette er en ganske stor programmerings jobb, så vi begynte å lete etter andre løsninger. Vi kom frem til at stemmekommunikasjon var veien å gå, siden dette ligner på kommunikasjon via walkie talkie. Bruken av stemmekommunikasjon gjør scenariet mer realistisk, siden personell på et skadested kommuniserer via walkie talkie i virkeligheten.

4 Implementering, koding og produksjon

4.1 Plattform

Oppdragsgiver stilte krav om at Dirus skulle kunne kjøres under Windows. Dette falt også for oss naturlig, da de fleste grafikkmotorer vi vurderte, støttet Windows. Det virtuelle spillbordet Dirus er utviklet i Microsoft Visual Studio .NET og fungerer bare på Windows 95/98/2000/XP. Dirus fungerer på Windows NT.

4.2 Utviklingsmiljø

I utgangspunktet hadde vi tenkt å basere oss på at skolen holdt oss med PC-er. Dette viste seg fort å være utilstrekkelig, da skolen kun hadde PentiumII300 og -233 maskiner til disposisjon, og ingen av disse hadde grafikkort med 3D-akselerasjon. Det endte derfor med at vi oppgraderte den ene skole-PC-en med mer RAM og grafikkort med 3D-akselerasjon, brukte en eldre Pentium3-PC fra ene prosjektdeltageren, samt at det ble innkjøpt en helt ny maskin. Den ene PC-en som skolen holdt ble brukt til CVS-server.

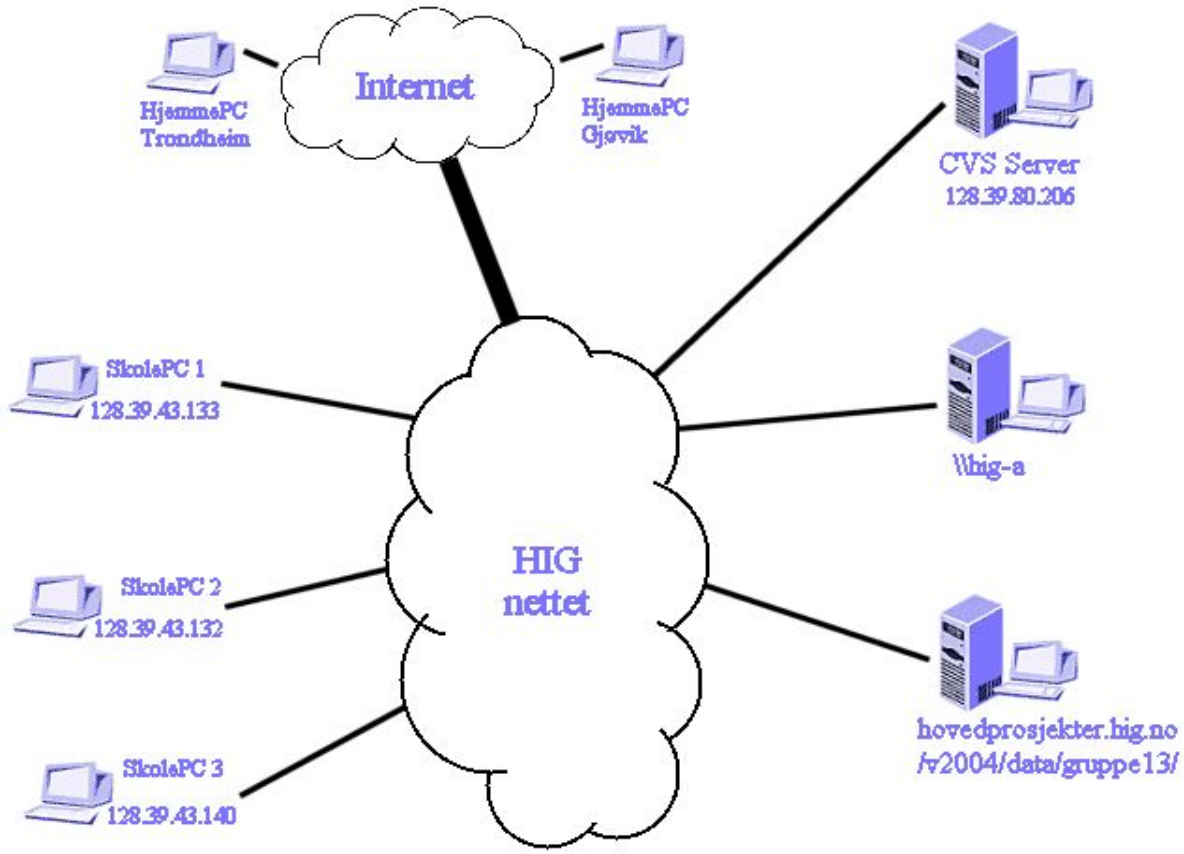
Programvare på arbeidsstasjonene:

- Microsoft Windows XP
- Microsoft Visual Studio .NET (VS7)
- Microsoft Project 2000
- OpenOffice v1.1
- Genesis 3D v1.1 motoren m/GTest, Genedit/Gedit, Actview osv.
- Adobe Photoshop v7.0
- Discreet 3D Studio Max 5.1
- Curious Labs Poser 5.0
- WinCVS v1.2 Poseidon for UML v2.1.2
- MilkShape 3D v1.7.0

Programvare på CVS-serveren:

- Debian Linux
- Linux CVS Server v.1.11.1p1
- CVS Repository for WEB
- FTP/SSH/WEB-servere

Klient-PC-ene har blitt brukt til programmering, modellering osv. Linux-maskinen er brukt som CVS-server for kildekode, rapporter, logger, hjemmeside for prosjektet osv.



Figur 20Nettverkstopologi i utviklingsmiljøet

4.3 Valg av verktøy

4.3.1 Valg av utviklingsverktøy

Programmeringsverktøy:
Microsoft Visual Studio
Før prosjektet startet, hadde flere av gruppemedlemmene lyst til å bruke java til utviklingen, men visual studio ble valgt, på grunn av at både Genesis3D motoren, samt GTest er skrevet i Visual Studio 6.0. Konverteringen til VS7 ble gjort automatisk første gang prosjektfilen ble åpnet. Av Visual Studio pakken, er det kun C/C++ delen vi har brukt.
Det var ingen store ulemper med at VS ble brukt i stedet for java, siden alle i gruppa fra før hadde programmert i både C og C++.

Konfigurasjonsstyring:
Concurrent Versions System (CVS)
CVS er et verktøy som sørger for at alle har tilgang til siste versjon av alle filer som det jobbes med i et prosjekt. Det gjør det enkelt å ta backup og gjør gruppemedlemmene geografisk uavhengige. CVS Repositotry gjør at man enkelt kan se alle filene som ligger på CVS-serveren, historiske forandringer, hvem som har gjort forandringene, kommentarer til forandringene i en hvilken som helst web-browser. I tillegg gir den enkel mulighet for å gå tilbake til en tidligere versjon av filene, å se hva slags forandringer som har blitt gjort på en enkel måte.

3D-Modelleringsverktøy:

- Genedit/GEdit: Brukes til modellering av brett for Genesis3D applikasjoner. Dette programmet har et grensesnitt som ligner på 3DStudio Max, noe som forenkler bruk for personer som alt kan 3Dstudio Max.
- 3D Studio Max: Er et program som ofte brukes for utvikling av 3D-modeller. Det finnes en plugin for eksportering til genesis3D actor filer siden 3Dstudio ikke har dette som standard, noe som gjør det relativt enkelt å lage nye actorer som vi kan bruke.
- Curious Labs Poser v5.0: Bra modelleringsverktøy for å gjøre modellering av personer enklere. Støtter eksportering til 3DStudio Max for videre bearbeiding.
- MilkShape 3D v1.7.0er et modelleringsverktøy som er utviklet spesielt for spill utviklere på hobbynivå. Det har støtte for de fleste grafikkmotorene lisensiert under LGPL, deriblant Genesis3D. Vi har benyttet MilkShape til å animere bevegelser på personellet i Dirus.

4.4 Kodestandard

Siden produktet vi står igjen med etter fullført prosjekt stort sett består av kode vi ikke har skrevet selv, var det naturlig at vi brukte samme kodestandard som kildekoden til GTest. Dette er også gjort i annen kildekode vi har funnet på nettet som har modifisert Genesis3D/GTest. Det virker som om Genesis3D-utviklerne har en slags DeFacto-standard på hvordan ting skal skrives. Vi har i tillegg til dette prøvd å ha gode rutiner på kommentering av kode vi har skrevet/modifisert, slik at det er enkelt å se hva som er nytt i forhold til originale Genesis/GTest.

De aller fleste hovedprosjekter på datalinjen, starter gjerne med blanke ark. Vi derimot begynte med 200 000 linjer med kode, 150 000 i Genesis3D og 50 000 i GTest. Det å åpne GTest kildekoden for første gang var en smule overveldende, da ingen av oss hadde skrevet noe i nærheten av noe så stort før. Faktisk var det ikke noe problemer og komme igang og få kompilert GTest koden for første gang. Det som derimot var den største utfordringen var å vite hvor en skulle begynne for å legge til/forandre GTest så det skulle bli slik vi ønsket den skulle bli. Som nevnt tidligere, var en av grunnene til at vi valgte Genesis3D som plattform, at miljøet rundt motoren var stort. Bl.a. er det mange sider på nettet som er spesielt egnet for personer som driver med Genesis3D/GTest modifisering:

http://www.genesis3d.com/
http://www.genesis3d.com/~kdtop
http://www.genesis3d.com/~wog/
http://www.genesis3d.com/forum/

Spesielt de to siste linkene har vært nyttige som oppslagsverk, og forumet har blitt benyttet flittig til spørsmålsstilling.

4.5 Eksempler på kildekode

4.5.1 Eksempel 1

Her forklares i detalj hvordan en KA kontinuerlig sjekker om et flagg er satt ut av kontrollør, og på dette grunnlag avgjør KA-ens bevegelse.

Funksjonen `Bot_Modethink` er en funksjon i filen `bot.c` som simulerer hva KA-en til enhver tid tenker. Denne funksjonen startes med korte intervaller, omtrent tre pr. sekund. Det er derfor en nyttig funksjon til å implementere løsninger til KA-ens AI som krever kort respons. Funksjonen kjøres av alle KA-er i brettet som spilles.

Funksjonen `Bot_Modethink` startes med tre medsendte variable:

- `VSI`: Aktiv brett, er alltid den samme for hvert spill som er startet.
- `PlayerData`: Genesis3D bruker `Player` om aktører. `Playerdata` angir hvilken aktør som skal modifiseres, i dette tilfelle en KA. Inneholder datastrukturen til aktuell KA, som f.eks. posisjon, fart osv.
- `Time`: Benyttes til andre formål, ikke relevant her.

```
geBoolean Bot_ModeThink(GenVSI *VSI, void *PlayerData, float Time)
{
```

De medsendte datastrukturer legges i egne variable.

```
GPlayer           *Player,*Hit;
Bot_Var           *DBot;
Player = (GPlayer*)PlayerData;
DBot = Player->userData;
```

Hvis motoren er satt til å skrive ut debug-setninger, vil informasjon om KA-ens modus skrives ut på skjerm

```
if (BotDebugPrint) GenVSI_ConsoleHeaderPrintf(VSI, DBot->TgtPlayer-
>ClientHandle, GE_TRUE, "Bot Think Mode %s", ModeText[DBot->Mode]);
```

Hjelpefunksjon til Bot_ModeThink, ikke relevant her.

```
Bot_ModeThinkFunc[DBot->Mode](VSI, PlayerData, Time);
```

Her kjøres en uendelig løkke, som kun avbrytes hvis Dirus ikke finner noen aktører i brettet.

```
Hit = NULL;
while (1)
      {
```

Hvis funksjonen GetNextPlayer finner en aktør, legges datastrukturen til denne aktøren i variablen Hit.

```
Hit = GenVSI_GetNextPlayer(VSI, Hit, NULL);
```

Ingen aktør funnet, bryter ut av løkken.

```
if (!Hit)  break;
```

Sjekker om Hit er et SAPL Evakuerings-flagg og om aktuell KA er publikum.

```
if (Hit->ViewIndex == ACTOR_INDEX_SAPLEVAK && Player->ViewIndex  ==
ACTOR_INDEX_PLAYER)
            {
            Setter aktuell KA til å gå mot posisjonen til Hit.(flagget)
            DBot->GoalPos = Hit->Pos;
```

```
                Bot_InitMoveCloser(VSI, Player, 10.0f);

                Hvis KA-en har nådd frem til flagget, (KA og flagg har samme posisjon)
                        stoppes KA-en.
                if (Bot_PastPoint(&Player->XForm.Translation,&DBot-
                >MoveVec, &DBot->TgtPos))
                        {
                        DBot->RunSpeed = 0.0f;
                        }
                }

                Bot er av typen skadd publikum, og har funnet SAPLSKADDE-flagg.
                if (Hit->ViewIndex == ACTOR_INDEX_SAPLSKADDE && Player-
                >ViewIndex==ACTOR_INDEX_SKADDE)
                        {
                        Går mot flagget
                        DBot->GoalPos = Hit->Pos;
                        Bot_InitMoveCloser(VSI, Player, 10.0f);

                        Stopper hvis den er ved flagg
                        if (Bot_PastPoint(&Player->XForm.Translation, &DBot-
                        >MoveVec, &DBot->TgtPos)) DBot->RunSpeed=0.0f;
                        }
                }
        return GE_TRUE;
}
```

4.5.2 Eksempel 2

Statusbar er et felt nederst på skjermen i Dirus. Den er tredelt og gir brukeren informasjon om aktuelle taster, verktøy som er valgt, samt informasjon i løpet av spillets gang.
Her er noen viktige utdrag fra hvordan dette er løst i kildekoden vår, med vekt på det midterste feltet for verktøy.

- *client.c*

Client_CreateStatusBar()

Laster inn bitmaper (bilder) som skal vises på statusbaren. Funksjonen laster et bitmap for hver del av statusbaren.

```
for (i=0; i<NUM_OF_ELEMENTS; i++)
{
  for (k=0; k<3; k++)
  {
    char  Name[64];
    sprintf(Name, "Bmp\\SBar\\%s\\DSBar%i-%i.Bmp", BmpPath, i, k);
    assert(Client->StatusBar.Elements[i].Bitmaps[k] == NULL);
    Client->StatusBar.Elements[i].Bitmaps[k] =
geBitmap_CreateFromFileName(MainFS, Name);
  }
}
```

Legger hver enkelt bitmap inn i en array. Denne arrayen inneholder bitmaper til alle verktøy .
Den setter så fargepalettens høyeste verdi (255) til å være transparent.

```
DirusBitmaps[0]=geBitmap_CreateFromFileName
(MainFS,"Bmp\\Dirus\\sklko.bmp")
DirusBitmaps[1] = geBitmap_CreateFromFileName
(MainFS,"Bmp\\Dirus\\saplskadde.bmp");
DirusBitmaps[2] = geBitmap_CreateFromFileName
(MainFS,"Bmp\\Dirus\\saplevak.bmp");
DirusBitmaps[3] = geBitmap_CreateFromFileName
(MainFS,"Bmp\\Dirus\\ventamb.bmp");
DirusBitmaps[4] = geBitmap_CreateFromFileName
(MainFS,"Bmp\\Dirus\\hel.bmp");
DirusBitmaps[5] = geBitmap_CreateFromFileName
(MainFS,"Bmp\\Dirus\\akp.bmp");
DirusBitmaps[9] = geBitmap_CreateFromFileName
(MainFS,"Bmp\\Dirus\\sperrebukk.bmp");
for (i=0;i<=9;i++)
{
      if (DirusBitmaps[i])
      {
      geBitmap_SetColorKey(DirusBitmaps[i], GE_TRUE, 255, GE_FALSE);
            geEngine_AddBitmap(Client->Engine, DirusBitmaps[i]);
      }
}
```

DrawStatusBar()
Når et verktøy er valgt, tegnes et bitmap som representerer verktøyet i statusbarens midterste del. De to siste parametrene i metoden geEngine_DrawBitmap() angir posisjonen til bitmapet.

```
if (i == Client->CurrentWeapon) {
      geEngine_DrawBitmap(Client->Engine, DirusBitmaps[i], NULL, 228,
422);
}
```

For å se antall verktøy tilgjengelig for spilleren.

```
if (i == Client->CurrentWeapon) {
      Val[i] = Ammo;
      sprintf(Temp, "%i", Val[i]);
      Console_XYPrintf(GameMgr_GetConsole(Client->GMgr), 300, 430, 1, "%s",
Temp);
}
```

- *console.c*

Bytte ut de opprinnelige våpnene i GTest med flagg.

Metodene `FireBlaster()`, `FireGrenade()`, `FireRocket()` og `FireShredder()` ble endret til å ligne en modifisert versjon av `FireRocket()`, i tillegg ble det lagt til funksjoner for HEL(Helikopter landingsplass), AKP (Ambulansekontrollpunkt) og Sperrebukk. Her er noen viktige utdrag fra en funksjonene:

```
void FireRocket(GenVSI *VSI, void *PlayerData, float Time)
{
      (...)
```

Verktøyene har visse parametere som settes for å vise hvordan de skal vises for spilleren.

```
      Weapon->ViewFlags = (VIEW_TYPE_ACTOR);
```

Hva actoren heter i motoren. Definert i `gamin.h`

```
      Weapon->ViewIndex = ACTOR_INDEX_SAPLEVAK;
```

Skalerer størrelsen på verktøyet

```
      Weapon->Scale = 3.0f;
```

Våpenet skal i skytes ut med samme transformasjonsmatrise som spilleren.

```
      Weapon->XForm = Player->XForm;
```

Justerer flagget til å stå rett opp.

```
      Weapon->XForm.BZ=0;
```

Finner ut hvor flagget skal plasseres i forhold til spillerens posisjon. Det kan justeres i x-, y- og z-retninger i forhold til spilleren.

positiv: Forover, negativ: bakover

```
      geVec3d_AddScaled(&Front, &In, 600.0f, &Front);
```

positiv: Oppover , negativ: nedover

```
      geVec3d_AddScaled(&Front, &Up,   0.0f, &Front);
```

positiv: Høyre, negativ: venstre

```
geVec3d_AddScaled(&Front, &Rt,   0.0f, &Front);
```

For å gi tilbakemelding til spilleren om at vi har satt ut et verktøy, skrives det ut på statusdelen av statusbar:

```
GenVSI_ConsoleHeaderPrintf(VSI,Player->ClientHandle,GE_TRUE,"SAPL
Evakuerte satt ut");

(...)
}
```

`Control()` metoden

```
geBoolean Rocket_Control(GenVSI *VSI, void *PlayerData, float Time)
{
(...)
```

Med flagg skal det bare settes ut en actor, og den skal stå der til spillets slutt. Setter derfor telleren til å hele tiden være satt til null.

```
Speed = Player->Time*0.0f;

(...)
}
```

Nye `ACTOR_INDEX-er` (forskjellige aktører) må defineres i `gmain.h`.

F.eks. `ACTOR_INDEX_SAPLSKADDE` er referansen motoren bruker til denne spesifikke actoren. Disse får et unikt index-nummer. Her er et utdrag av noen `ACTOR_INDEX-er`:

```
#define ACTOR_INDEX_SAPLSKADDE      15
#define ACTOR_INDEX_SAPLEVAK  16
#define ACTOR_INDEX_BUKK            19
```

Og nye Actor-filer må lastes i `SetupWorldCB() i level.c`.

```
GenVSI_ActorIndex(VSI, ACTOR_INDEX_SAPLSKADDE, "foo",
"Actors\\saplskadd.act");
```

4.5.3 Eksempel 3

Her forklares hvordan vi implementerte "OverheadCam," hvor kontrolløren får et fugleperspektiv-synsfelt. Se

- `gmain.c:`

Global variabel som angir om OverheadCam er aktivert. Settes til GE_FALSE, slik at man starter spillet med vanlig synsfelt.

```
geBoolean OverheadCam = GE_FALSE;
```

Setter setter variabelen nevnt ovenfor til GE_TRUE hvis brukeren trykker «J»-tasten.

```
if (IsKeyDown('J'))
	{
		OverheadCam=!OverheadCam;
		return GE_TRUE;
	}
```

- `client.c`

Henter global variabel fra `gmain.c`, nevnt ovenfor.

```
extern geBoolean OverheadCam;

if (!OverheadCam)
	{
```

Hindrer at x-aksen ikke forandres når brukeren skifter til OverheadCam.

```
	if (MouseInvert)
	Client->Angles.X -= GlobalMouseSpeedY;
	else
	Client->Angles.X += GlobalMouseSpeedY;
	}


if(!b1stPerson || OverheadCam) {
```

Setter motoren til å rendre spilleren hvis man er i OverheadCam-modus. Sender også feilmelding hvis motoren får problemer.

```
if(geWorld_AddActor(World, Player->Actor, GE_ACTOR_RENDER_NORMAL|
GE_ACTOR_COLLIDE, 0xffffffff) == GE_FALSE)
		GenVS_Error("[CLIENT] CheckClientPlayerChanges:  Could not
add actor to world.  ActorDef Name: %s.\n", ActorIndex->FileName);
}
```

Setter verdier for kamera-vinkelen for både vanlig perspektiv og OverheadCam-perspektiv.

```
if (OverheadCam)
		{
		camAngle = -1.57; \\Pi/2
		camHeight = 800.0f;
		}

		else
		{
		camAngle = 0.0f;
		camHeight = 140.0f;
		}
```

Plasserer kameraet i forhold til spilleren.

```
geXForm3d_RotateX(&Client->ViewXForm, Client->Angles.X + camAngle);
Client->ViewXForm.Translation.Y += camHeight;
```

5 Testing og kvalitetssikring

5.1 Kvalitetssikring

5.1.1 Sikring av data og backup

Gruppen var tidlig klar over farene ved tap av data. Ingenting er mer demotiverende enn å miste flere ukers arbeid. Vi bestemte oss derfor tidlig å utnevne en egen person på gruppa som hadde ansvar for å ta backup av alle data en gang i uken. At vi lagrer all data på CVS-serveren, gjør det enkelt å ta backup, og gjorde oppdaterte versjoner ble tilgjengelig for hele gruppa. Vi fikk etterhvert en rutine på å sende alle endrede data til CVS med en gang endringene ble utført. Dette virket bra, inntil en dag da vi ikke lenger fikk startet serveren med CVS. Dette skapte problemer for oss: Alle satt plutselig med utdatert kildekode, og når forandringer i kildekode ble gjort, kunne ikke disse legges inn på CVS-serveren. Ved en feil ble backup-filene plassert på samme disk som CVS-serveren kjørte på. Siden vi ikke fikk startet denne, fikk vi heller ikke gjenopprettet backupen (på dette tidspunktet var det uklart om det var disk-krasj vi hadde hatt, eller om det bare var problemer med selv oppstarts-sekvensen). Vi fikk etter hvert startet CVS-serveren med Knoppix (en linux-distribusjon på CD), og det viste seg heldigvis at våre data ikke ble skadet. Vi fikk hjelp av en av skolens ansatte, Erik Hjelmås, og det viste seg at det problemet vi hadde var en kjent feil. Vi fikk reinstallert operativsystemet, satt igang CVS-serveren, og gjenopprettet all data.

Disse problemene førte til at vi gjorde visse endringer på rutinene for backup:

- Backup blir nå overført via FTP til en annen maskin av backupansvarlig.
- Antall backup per uke er doblet.
- Backup nr. 2 brennes ut på CD, slik at hvis en hendelse ødelegger backup nr 1, samt alle uthentede versjoner fra CVS-serveren, har vi fortsatt en backup et annet fysisk sted.

5.2 Testing

5.2.1 Testing av kildekode

Testing av kildekoden ble utført av gruppens medlemmer etter hvert som prosjektet skred frem. Metoden vi brukte var “empirisk testing”, som på godt norsk kan kalles «prøve og feile»-metoden. Det betyr rett og slett at når vi la inn ny kode eller modifiserte kode testet vi om det virket som det skulle. Hvis ikke, ble det rettet opp. Noe som skjedde ved flere anledninger, var at utvikleren ikke oppdaget feilen selv. Da oppdaget alltid noen andre på gruppa problemet. Bl.a. var dette tilfelle med flagg-systemet. Man har en viss mengde flagg tilgjengelig. Mengden ligger i en array. Siden koden for alle flaggene var kopiert fra en plass, var alle variabelnavnene like. Dette førte til at feil flagg-mengde ble telt ned etter hvert som flagg ble satt ut. Dette førte til at det så ut som en hadde igjen flagg av en type, men man fikk ikke satt det ut siden den sjekka på et annet flagg sin mengde.

5.2.2 Testing av 3D-modeller

3D-modellene, starter enten sitt liv i Poser, for så å bli eksportert til 3Dstudio, eller blir laget rett i 3DStudio. For å eksportere fra 3DStudio, er det et eget plugin man må bruke. Problemet med 3DStudio og Genesis3D, er at 3DStudio gjerne vil koble en vertex til flere bones, mens Genesis3D kun klarer at en vertex er koblet til ett bone. Det betyr at når vi prøver å bygge Actoren med Actorbuild, så får en error. Dette kan ordnes i 3DStudio før eksportering. Ellers så testes enten Actor-fila i Actorview for å se at den ser grei ut, eller den kan testes i et map som kjøres i Dirus.

5.2.3 Testing av kunstige aktører (KA-er)

Testing av hvordan KA-ene oppfører seg var ganske enkelt, da det er en switch i bot.c som heter "BotDebugPrint". Ved å sette denne til "TRUE", vil man få skrevet ut på skjermen hva KA-ene tenker, altså hvilken modus de er i til enhver tid. Etterhvert utvidet vi "BotDebugPrint" til å skrive ut flere variable, bl.a. hvilken type punkt en KA befant seg på, f.eks. "PointWaitForPlayer." BotDebugPrint ble også brukt til testing av iterasjoner, posisjonstester og -avgjørelser i kildekoden. Dette gjorde at vi fikk mer kontroll på testingen, og gjorde den lettere å utføre.

6 Installasjon/realisering

6.1 Introduksjon

Da målgruppen til Dirus ikke nødvendigvis har noe særlig dataerfaring, ble det for oss viktig å realisere en installasjon av Dirus så enkelt som mulig for brukerne. Dette innebærer at alle med grunnleggende datakunnskaper kan installere og starte Dirus, dessuten foreta enkelt vedlikehold. Dette er viktig da Dirus er ment å kjøre på flere heterogene systemer, uten at brukeren behøver å tilpasse Dirus til de forskjellige systemene.

6.2 Installasjon

Vi fant det derfor hensiktsmessig å legge Dirus i sin kjørbare form på en CD-ROM-plate, uten noen form for komprimering eller automatisk installasjon. Dette innebærer at brukerne kan velge mellom å kjøre Dirus direkte fra CD-platen eller kopiere CD-platens innhold på harddisk og kjøre Dirus derfra. For brukerne vil kopiering til harddisk medføre økt hastighet, da harddisker har høyere lesehastighet enn CD-ROM-spillere. Denne løsningen har flere fordeler, først og fremst sin enkle og lettforståelige form; en installasjon av Dirus vil foregå raskt og enkelt. Installasjonfilen for applikasjonen for stemmekommunikasjon over nett, Ventrilo, ligger også på CD-platen.

6.3 Vedlikehold

Vil man fjerne Dirus fra systemet, kan man bare slette Dirus-katalogen, da Dirus ikke gjør endringer i Windows' register. Dette gjør også vedlikeholdsarbeid enklere, skulle for eksempel en fil være skadet eller fjerne ved et uhell, kan man bare kopiere innholdet på CD-platen inn på harddisken på nytt. Innholdet på CD-platen vil være den eksekverbare filen til Dirus, dessuten diverse binærfiler som brukes av Dirus. Vi har også valgt å legge designdokumentet på CD-platen, dette for å få brukerne til å få et litt mer helhetlig inntrykk av hele systemet.

6.4 Andre applikasjoner

Da det har kommet ønske fra oppdragsgiver om å ha muligheten til å endre brettet ved senere anledning, har vi også lagt ved selve kart-editoren og prosjektfilen til kartet. Dette innebærer at oppdragsgiver kan lage nye brett eller endre det eksisterende brettet mhp. antall aktører, plassering til kjøretøy, branner, personell osv. Til stemmekommunikasjon mellom aktørene bruker vi gratisprogrammet Ventrilo.

7 Beskrivelse av utviklingsprosessen

7.1 Om Inkrementell systemutvikling i Dirus

7.1.1 Definere rammekrav

Rammekravene ble definert av oss i kravspesifikasjonen, og er basert på sivilforsvarets oppgavebeskrivelse. I utgangspunktet hadde vi tre rammekrav som vi definerte i kravspesifikasjonen, disse er :

- Forskjellige modus: dvs. hvem som er server/klient i spillet, hvilke rettigheter de forskjellige spillertypene har osv.
- GUI : Grafisk grensesnitt mellom bruker og systemet, f.eks. menyer, statusbar.
- Scenario: Design av selve brettet det spilles på, teksturer, actorer, effekter osv.
- System: Annen funksjonalitet knyttet til grafikkmotoren.

7.1.2 Fordele rammekrav til inkrementer

De forskjellige rammekravene ble fordelt til inkrementer etter denne tabellen

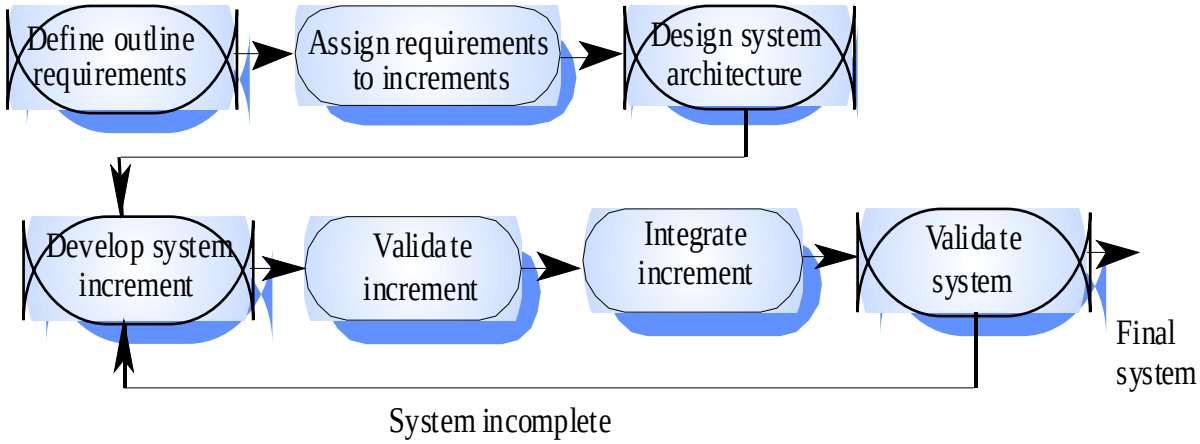
Inkrement navn	Fra rammekrav
Console	GUI
Endring av perspektiv	GUI
Specific Actor	System
Kunstige Aktører (KA)	System
Hovedmenyen	GUI
Effectsmanager	Scenario/System
DistanceFog, FarClipping, Ytelsesforbedring	System
Lage modeller/actorer	Scenario
Loading-bilde	System
Statusbar-modifisering	GUI
Flaggsystemet	GUI
Actorvariasjon blant spillere	Forskjellig modus
Pausefunksjon	GUI
Stemme-Kommunikasjon	Eksternt program
Lage brett	Scenario

Figur 21 Tabell med de forskjellige inkrementene

Inkrementene ble utført fortløpende etter denne listen. De nederste inkrementene er de som ikke var så viktige, som var lette å utføre til slutt, eller som rett og slett måtte utføres på slutten p.g.a. at tidligere inkrementer måtte på plass for at de skulle være mulig å lage.

Grunnen til at det ble så mange inkrementer i forhold til rammekrav, var at det er enklere å dele opp i mange enkle inkrementer, i stedet for veldig store som en kanskje ikke blir

ferdig med. I ettertid kunne vi kanskje ha fordelt "Lage Modeller/actorer"-delen bedre, siden det viste seg at denne tok veldig lang tid å fullføre, spesielt animering av actorer. Som en ser ut fra denne listen var GUI-delen en veldig stor bit av prosjektet sett i forhold til antall inkrement.



Figur 22Illustrasjon av Inkrementell Systemutvikling

7.1.3 Design av system arkitekturen

I selve systemet er rammeverket Genesis3D/GTest. Dette var i utgangspunktet ferdig designet når vi tok det i bruk, så vi trengte bare å utvikle og implementere inkrementene vi hadde definert, det var ikke nødvendig å designe datastruktur o.l.

7.1.4 Utvikle system inkrement

I startfasen av et programmerings-inkrement så gikk vi igjennom alle krav til dette inkrementet som var beskrevet i kravspesifikasjonen. Deretter gikk vi gjennom kildekoden til GTest og Genesis3D for å se om det allerede fantes moduler eller funksjoner som kunne brukes eller modifiseres. Vi gikk også gjennom Genesis forumet og andre ressurser på Internett forbundet med Genesis3D, for å få informasjon om hvordan ting burde løses. Oftest så utviklet vi ikke helt nye enkeltstående moduler, men modifiserte eller utvidet koden som allerede fantes. Vi jobbet her mest etter «prøv og feil» metoden, og testet etterhvert som vi la til funksjonalitet. Når et programmerings-inkrement var ferdig så ble det skrevet et eget dokument som dokumenterte alt som ble gjort av koding i det inkrementet

I startfasen av et inkrement som tok for seg utvikling av scenariet, så ble også kravspesifikasjonen gjennomgått. Her hadde vi stort sett krav til hvilke elementer scenariet skulle inneholde, men kravene var ikke veldig detaljerte. Grunnen til dette var at det var vanskelige å definere noen detaljerte krav til figurer og teksturer før man ble litt mer kjent med grafikkmotoren. Derfor ble detaljene utarbeidet i begynnelsen av hvert inkrement før vi begynte å utvikle deler av scenariet.

Her kunne vi se på hvert inkrement som en egen modul og disse kunne integreres i scenariet når de var ferdig utviklet. Et eksempel på dette er utviklingen av hvilke roller de ulike KA-ene skulle ha. Vi implementerte først en generell AI felles for alle KA-er. Etter videre konsultasjon med oppdragsgiver, endret vi AI-en til hver enkelt type KA, og ga de forskjellige roller som politi, skadde, brannpersonell, skadde osv.

7.1.5 Validere inkrementer

Validering av de forskjellige inkrementene ble gjort på forskjellige måter. Programmerings-inkrementene ble validert når de var fullt integrert og testet. Scenario inkrementene ble testet og validert når før de ble fullt integrert i scenariet, siden disse var selvstendige moduler.

7.1.6 Integrering av inkrementer

Programmerings-inkrementene ble, som nevnt i avsnitt **7.1.4**, så modifiserte eller utvidet vi kildekoden og utviklet sjelden helt enkeltstående moduler som skulle integreres. På denne måten ble de enkelte inkrementene integrert fortløpende mens vi kodet. De få programmerings-inkrementene som kunne betraktes som en egen modul, f.eks. Effectsmanager, ble integrert i systemet og testet etter at de var ferdig kodet.
Scenario inkrementene ble alle utviklet ferdig før de ble integrert og testet i landskapet,som var det første scenario inkrementet.

7.1.7 Validering av systemet

Validering av hele systemet ble i stor grad utført samtidig med validering av det enkelte inkrement. Dette gikk ut på å se om de andre delene av Dirus fortsatt fungerer som de skulle etter at et nytt inkrement var lagt til.

7.2 Vår anvendelse av Inkrementell systemutvikling

Vi fordelte oppgavene slik at en person stort sett jobbet alene på ett inkrement til det var ferdig validert inn i Dirus. Vi satte heller ikke noe krav om at et inkrement måtte være ferdig før en begynte på neste. Dette kunne være en litt risikabel strategi, siden vi risikerte å sitte med masse halvferdige biter når prosjektet gikk mot slutten. Men vi har stolt på alles interesse i at prosjektet ble ferdig i tide, og at det ble så bra som mulig. Det viste seg at dette ikke ble noe problem.

Fordelen med å kunne jobbe med flere inkrementer samtidig, er at hvis en føler at en står fast og ikke får gjort noe, er det like greit å begynne på noe annet. Når man så senere ser på inkrementet igjen , kan det hende man har fått nye kunnskaper og ideer om en fornuftig løsning.

Dette var bl.a. tilfellet når det gjaldt usynlighet for kontrolløren. Siden dette var en viktig bit av prosjektet, begynte vi relativt tidlig med å løse denne. Alle actorer må ha en viss variabel satt for at den skal rendres av motoren. Tanken var da at man ved å ikke sette denne variabelen, kunne slå av rendring av den actoren som var kontrollør. Etter 2-3 dager der en på gruppa kun hadde hatt fokus på dette, måtte vi se slaget som tapt. Andre inkrementer ble jobbet på og utviklet i mellomtiden, hvor vi etter ca. 2 måneder igjen så på dette problemet. Da var kunnskapene om Genesis3D/GTest så mye større at løsningen ble implementert på en halv dag.
Det viste seg at tanken bak den opprinnelige implementasjonen var riktig, mens vi endret på feil kildekode. I tillegg måtte vi dypere ned i kildekoden, og jobbe med include-filer, noe vi ikke var fortrolig med i begynnelsen.

8 Diskusjon av resultater

8.1 Resultater

Vi skulle gjerne ha likt å si at hele prosjekt Dirus ble bra, perfekt, strålende, men det ble det ikke. Selv om vi har nedlagt nesten all vår tid i løpet av året i hovedprosjektet, har det vært andre fag, eksamener og prosjekter som har krevd sitt av tid. Vi vil allikevel si at vi har nedlagt all ledig tid i dette prosjektet. Noe ble vi meget fornøyd med og er løst bedre enn vi hadde forventet, mens andre ting er løst dårligere enn vi hadde håpt vi skulle klare, eller rett og slett ikke løst fordi vi ikke fant en løsning, eller fordi tiden ikke strakk til.
Vi er allikevel stolte av det vi har fått til, og mener jobben er vel utført!

8.1.1 Hva som ble bra

- OverheadCam: Var ment som en mulighet for å se spilleren rett ovenfra, men utviklet seg til å bli en mulighet til å se seg rundt på hele brettet. Via OverheadCam kan man stå på hvilket som helst punkt og se hele scenariet. Dette gir meget god oversikt over alt som foregår.
- Flagg m/overliggende plan: Vårt «brainchild», hvordan flaggene skulle få en fornuftig dimensjon på bakkenivå, samtidig som de skulle være lesbart fra langt ovenfra. Vi løste dette ved å lage en flate som kun ble rendret på den ene siden. Vi hektet en slik flate på alle flaggene slik at flatene hang høyt i luften rett over flagget. Den siden som blir rendret er rettet oppover mot himmelen, og på denne har samme tekst som flagget. På den måten går det an å lese flagget via OverheadCam ved å se på disse flatene, mens fra bakken så er de usynlige.
- Kunstige aktører: De kunstige aktørene var noe vi var usikre på om vi kom til å klare å gjøre noe bra ut av. Vi er godt fornøyd med at vi kan tildele de kunstige aktørene forskjellige roller i et scenario, og at de forskjellige rollene reagerer på forskjellige flagg.
- Statusbar: Vi synes vi har utviklet en bra statusbar. Den er lett å bruke og gir brukeren det meste av informasjon en trenger i et scenario.
- Effekter: Effektene våre er vi også godt fornøyd med. De er lette å plassere ut i scenariet og man har mange valgmuligheter når en skal bestemme omfang, retning, hastighet og plassering av disse effektene. Det eneste som mangler er eksplosjoner.
- SpecificActor: Denne entiteten har gjort at vi kan sette sammen detaljerte scenarier uten at det krever altfor mye av systemet. Det eneste som mangler her er kollisjons deteksjon (dette blir nevnt nedenfor).

8.1.2 Hva som ikke ble så bra

- Tak på bygninger: Vi hadde problemer med at tak på bygningene ble «gjennomsiktige» på avstand. Dette ble delvis fikset på ved å lage taket som en tykkere boks, men det hender fortsatt at en ser gjennom tak og andre ting som er forholdsvis tynne.
- Veggtexture: Utendørs følelsen er kun en illusjon fra vår side. Egentlig er alt en «innendørs» boks, som har fått textures for å se ut som himmel og gress. Siden denne boksen er forholdsvis høy, blir textureing en litt merkelig affære. Siden maksimum tekstur størrelse er 256x256 piksler, må den strekkes ganske kraftig for å få veggen til å bli slik vi ønsker (grønn nederst, blå oppover). Det virker, men ser ikke helt naturlig ut.

8.1.3 Ytelseskrav versus utseende

Som nevnt tidligere i rapporten, krever Dirus endel ytelse. Dette betyr at vi har måttet spare inn hastighet der det har vært mulig.

- Lowpoly actorer: Her har vi gjort en avveining mellom hva som skal se pent ut og hva som skal være funksjonelt. Vi kunne fått veldig pene actorer i brettet ved å laget de med mange polygoner. Dette er spesielt gjeldende på runde hjørner på f.eks. biler. Mennesker har jo ganske runde kanter de også. For at ytelsen skal bli bra, har vi gjort mennesker og biler mer firkantet.
- Farcliping: For å øke ytelsen, kutter den rendring av ting som er en viss avstand fra spilleren. Denne klippinga foregår i en sylinderform rundt spilleren. En mer ideell form hadde vært en kule for at det skulle sett penere ut.

8.1.4 Hva som ikke virker

- Textpromt: En ting som var tanken fra starten av Dirus utviklingen, var at en skulle kunne få informasjon presentert som tekst ved å nærme seg en Kunstig Aktør vi hadde plassert ut. For at dette skulle virke, måtte TextPromt «kobles» på en actor. TextPromt var ikke laget for dette, men for å kobles på modeller. Samtidig hadde vi et problem med at hendelser som foregikk i scenarioet stoppet opp når TextPromt var i brukt. Disse to problemene klarte vi ikke å gå rundt.
- Kollisjons deteksjon mot actorer: Dirus har innebygget kollisjons deteksjon for actorer mot modeller, mens ikke actorer mot actorer. Dette har vi prøvd å implementere, men uten hell. Vi registrerer kollisjoner, men har ikke klart å blokkere actoren som løper inni en annen actor.

8.2 Muligheter og valg underveis.

8.2.1 Valg av grafikkmotor

Å sette seg inn i en grafikkmotor er en tidkrevende prosess, vi så derfor tidlig at vi så fort som mulig måtte avgjøre hvilken grafikkmotor vi skulle bruke til utviklingen. Av oppgavebeskrivelsen fantes det visse krav til funksjonalitet som grafikkmotoren enten måtte støtte, alternativt være så fleksibel at vi kunne implementere ønsket funksjonalitet. Da vi bestemte oss for Genesis3D var det følgende egenskaper vi ga til grunn for valget: Det er en relativt gammel motor, og krever derfor bare moderat maskinkraft. Det finnes en mengde dokumentasjon og kildekode til motoren, både fra allerede utviklede applikasjoner, web-sider og diverse fora på Internett. Den er gratis, lisensiert under LGPL. Støtte for nettverkskommunikasjon. Genesis3D har noen ulemper, det mest fremtredende er at det er beregnet på innendørs-scenarier, ikke store utendørsscenarier som vi har i Dirus, noe som økte kravene til maskinkraft.
Ogre3d, Crystal Space 3D og RealityFactory (som er en videreutvikling av Genesis3D) var de andre motorene vi vurderte. Vi kom etterhvert frem til at valget sto mellom Genesis3D v1.1 og RealityFactory. Begge motorer hadde demonstrasjons-spill som vi kunne ta utgangspunkt i. Genesis3D v1.1 hadde Gtest og RealityFactory hadde en demo som het RealityFactory. Vi valgte å bruke Gtest som utgangspunkt , da denne hadde lavere krav til maskinkraft enn RealityFactory. Det viste seg senere i utviklingsprossessen at to viktige problemer oppsto:

- Da vi begynte å utvikle adferden til de Kunstige Aktørene, stilte vi spørsmål om GTest fortsatt var riktig valg, da RF har en mer lettforståelig implementasjon av styring av KA-er, noe som ba på problemer i GTest i form av større programmeringskunnskaper. RF stiller mindre krav til programmeringskunnskaper, det ville da ha medført mindre arbeidspress å benytte oss av denne.
- Genesis3D er i utgangspunktet designet for et innendørs miljø, mens vi skulle lage et utendørs miljø. Dette problemet gjaldt også RealityFactory, siden den bygger på Genesis3D. De første testene vi utførte på maskinvare vi satte som minimum var tilstrekkelig til å kjøre brettene som fulgte med GTest spillet. Selv de første store brettene vi laget for å teste gikk tålelig bra. Men det viste seg at når vi la til flere hus, biler og mennesker, så ble det i tyngste laget for minimums maskinen vår.

Det kunne se ut som vi hadde valgt feil grafikkmotor, men disse problemene kom vi rundt til slutt. Da fordelene oppveide ulempene, sto vi fast på valget av Genesis3D med GTest som utgangspunkt. Mer om valg av grafikkmotor finnes i Design kapitlet.

8.2.2 Utviklingsspråk og -miljø

Både Genesis3D og GTest er skrevet i C og de bruker begge MS Visual Studio .NET, vi stod derfor uten valg både til språk og utviklingsmiljø. Det finnes en versjon av GTest for utvikling i Borland-miljø, denne er imidlertid i en så tidlig utviklingsfase at vi valgte Visual Studio-utgaven.

8.2.3 Modelleringsverktøy

Valget av modelleringsverktøy falt naturlig i sammenheng med valg av grafikkmotor, da grafikkmotoren kun benytter seg av spesifikke formater. Til modellering av aktører ble Discreet 3DStudio Max benyttet, mens vi benyttet Gedit til modellering av scenario. Alternativt kunne vi brukt Caligari Truespace, men valget falt på 3DStudio på bakgrunn av gruppens erfaringer med 3DStudio.

8.2.4 Servermiljø

Valg av server-miljø falt tidlig på GNU/Linux, først og fremst på grunn av denne plattformen gir større stabilitet enn Windows. GNU/Linux er også lettere å konfigurere og mer egnet til kjøre tjenester i forhold til filtjening, backup og konfigurasjonsstyring.

8.2.5 Konfigurasjonsstyring

Til konfigurasjonsstyring valgte vi å kjøre Concurrent Versions System (CVS), da dette er det mest brukte gratisverktøyet for konfigurasjonsstyring. CVS er dessuten gratis å bruke. Et alternativ til CVS er Subversion, men da Subversion foreløbig bare er gitt ut som betaversjon, stod vi fast ved valget av CVS. Da ingen av oss hadde erfaring med CVS fra før, brukte vi WinCVS, som er et grafisk brukergrensesnitt til CVS.

8.2.6 Tekstbehandlingsverktøy

Til tekstbehandling valgte vi OpenOffice av flere grunner. OpenOffice er en gratis klone av Microsoft Word, Excel og Powerpoint, gruppens medlemmer var allerede godt kjent med disse, så ingen innføring i OpenOffice var nødvendig. Senere i prosessen fant vi også ut at OpenOffice er mer stabilt en Word, og har dessuten en mer effektiv lagringsbehandling, slik at filstørrelsen blir mindre enn ved Word. Dette var viktig for oss siden vi hadde begrenset med plass på CVS-serveren vår.

8.3 Alternative løsninger

En alternativ løsning hadde vært å satse på en annen type grafikkmotor. Oppdragsgiver så for seg en prototype som baserte seg på 1. persons skytespill. Av den grunn vurderte vi kun grafikkmotorer som var utviklet for slik bruk. I ettertid ser vi at en annen løsning hadde vært å bruke en grafikkmotor som var utviklet for spill som foregår i store utendørs landskap eller en strategispill. Med en slik motor hadde vi hatt mindre begrensninger av størrelsen på et scenario og vi hadde fått et mer realistisk utseende.
Et alternativ til løsningen med oppførselen til de kunstige aktørene hadde vært å bruke skript. På denne måten kunne man ha forandret oppførselen den de kunstige aktørene uten å forandre på koden.

8.3.1 Forslag til forbedringer og nye oppgaver

Forslag til forbedringer: Vi har brukt Genesis3D v1.1 . En forbedring ville vært å modifisere Dirus slik at den kan bruke Genesis3D v1.6. Denne versjonen har støtte for 32 bit textures. Dette vil føre til mer detaljert grafikk. Genesis3D v 1.6 er også tilpasset nyere Windows versjoner og er mer effektiv.
En annen forbedring er implementere kollisjons rutiner for actorer, slik at det ikke blir mulig å gå tvers igjennom sperrebukker o.l.
Det er potensiale for forbedringer når det gjelder animasjonene på figurene i Dirus. Det tar lang tid å animere, mer tid enn vi hadde, og vi føler at dette kunne vært gjort bedre.
Det burde legges til flere forskjellige scenarier.
Det burde legge til flere roller enn Skadestedsleder og Kontrollør. Eksempler på roller: fagleder brann, fagleder medisin, observatør o.l.
Implementere en stemmekommunikasjons-modul i Dirus, istedet for å bruke et eksternt program.
Forslag til nye oppgaver: en ny oppgave kunne vært å implementere alle forbedringene listet opp ovenfor, samt å implementere et system for skripting i tillegg.
En annen oppgave kunne være å utvikle en prototype basert på et strategi spill.

8.4 Evaluering av gruppas arbeid

8.4.1 Innledning

Når prosjektet startet var vi 4 personer som hadde hatt varierende kontakt med hverandre fra tidligere. To av gruppemedlemmene hadde jobbet sammen på prosjekter før, og dannet gruppens opprinnelige medlemmer, mens de 2 andre kom til «senere». Selv om alle 4 ikke hadde jobbet sammen tidligere, viste det seg at det ikke var noe stort problem å få gruppen til å dra i samme retning.

8.4.2 Organisering

Gruppen har hatt følgende delegerte ansvarsområder:

- Prosjektleder: Leder for prosjektet, delegerer arbeid, sørger for at ting blir gjort
- Kontaktperson: For veileder og for oppdragsgiver
- Loggansvarlig: Skriver logger for gruppemøter, veiledermøter og daglige logger.
- Modelleringsansvarlig: Ansvarlig for all modellering av 3D modeller og scenarier
- Server og Backup: Ansvarlig for CVS-serveren, og backup av denne.
- Web-ansvarlig: Ansvarlig for å oppdatere hjemmesiden med nye dokumenter og bilder.
- Programmerere: Programmere nye funksjoner og modifisere eksisterende->Dirus
- Hardware/Software: Ansvarlig for Hardware og Software installasjon

8.4.3 Fordeling av arbeid

Gruppeleder sto for fordeling av arbeidet. Personene som hadde størst kompetanse innen et felt fikk disse jobbene. Bl.a. gjaldt dette «Server og Backup», samt Modellering. Prosjektleder tok seg selv av å være kontaktperson for gruppa. En person ble ansvarlig kun for modellering. De resterende tre på gruppa måtte ta seg av å være programmerer siden det var her størsteparten av jobben lå. Disse tok seg også av de andre ansvarsområdene. Hardware/Software ansvarlig var kun en jobb som var aktuell de første ukene når arbeidsmiljøet til gruppen ble satt opp.

Da rapporten var ferdig, så loggen slik ut: 1500 timer totalt.

8.4.4 Prosjekt som arbeidsform

Alle på gruppen hadde jobbet med andre i prosjekter før, men ikke av denne størrelsen og det var en stor utfordring. Vi føler at vi har løst dette tilfredsstillende, men med rom for forbedring. Vi har tross alt lært veldig mye om hvordan en bør gjennomføre et prosjekt i løpet av dette semesteret. Bl.a. hvordan vi skal jobbe som en samlet gruppe, og bli enige om løsninger som skal implementeres.

8.4.5 Subjektiv opplevelse av hovedprosjekt

Det å jobbe med hovedprosjekt har vært kronen på verket etter 3 år ved HiG. Det har utfordret oss på mange områder, men vi har alltid beholdt en løs og ledig tone overfor hverandre, selv når vi har vært stresset. Det har vært perioder der motivasjonen ikke har vært på det høyeste, men vi har alltid bitt tennene sammen for å komme i mål med et best mulig resultat. Spesielt på slutten av prosjektet har det vært lange dager og sene kvelder med rapportskriving og fullføring av Dirus. Vi mener at dette prosjektet har gjort oss mer forberedt på hva som venter oss i arbeidslivet.

9 Konklusjon

Gjennom et halvt års arbeid har vi jobbet med et spennende og variert prosjekt, helt fra oppdragsgivers presentasjon av oppgaven til ferdigstillelse av Dirus og vår rapport. Prosjektet satte krav til samarbeid, planlegging og motivasjon i tillegg til faglige kunnskaper. Motivasjonen blant gruppens medlemmer har gjennom hele prosjektet holdt seg på et høyt nivå på bakgrunn av vår interesse for selve produktet vi har jobbet med. Vi har lært veldig mye om prosjektstyring gjennom dette prosjektet og ser på det som en nyttig erfaring.
Det var en ny erfaring å jobbe videre på et prosjekt som andre hadde startet (Gtest). Spesielt omfanget av kildekoden vi måtte studere i begynnelsen var nytt for oss. Genesis3D v1.1 inneholder 150 000 linjer med C kode, mens Gtest inneholder 50 000 linjer med C kode. Ingen på gruppen hadde noen gang jobbet med et så stort og komplisert programmerings prosjekt.
Vi ser også på dette som en veldig nyttig erfaring å ta med seg videre.

Vi sitter igjen med en oppfatning om å ha kommet frem til et produkt som tilfredsstiller oppdragsgivers krav, i tillegg til å ha tilfredsstilt våre forventninger til oppgaven. Oppgaven strakk seg over et halvt år, så tidsaspektet var spesielt for gruppens medlemmer. Hele prosessen har gitt oss flere allsidige og nyttige erfaringer som vil komme til nytte senere i livet. Selve oppgaven, sammen med Sivilforsvarets engasjement i prosjektet, har betydd mye for vår motivasjon. Vi er spesielt fornøyd med å kunne bistå Sivilforsvaret med deres opptrening av personell.

10 Litteraturliste

Eclipse Entertainment, 24.12.02, Home of Genesis: http://www.genesis3d.com/

BLAeQ8, World of Genesis:http://www.genesis3d.com/~wog/

OtherWorlds Interactive, 29.04.04: http://welcome.to/genesis3d-university/

kdtop, Unofficial Genesis3D API reference: http://www.genesis3d.com/~kdtop/

Jean Louis Clement, Genesis :http://www.jlstsi.com/

Seven, Genesis programming: http://www.genesis3d.com/~seven/projectz/projectz.html

Terry Morgan, 2003, Asylum : http://doors.cybertrails.com/~hike1/

DumbalumSoft, Milkshape 3D: http://www.swissquake.ch/chumbalum-soft/

Gekido Design Group, Reality Factory :http://www.realityfactory.ca/v3/

phpBB, Genesis3d Forum, http://www.genesis3d.com/forum/index.php

Jelsoft Enterprises Limited, Milkshape Forum, http://www.swissquake.ch/chumbalum-soft/forum/

Evolution Design, 3D resources, http://www.onnovanbraam.com/

Jimmy-4wd, Scratch made Cars, http://smcars.nd4spdworld.com/

3Dluvr, Where art an technology meet :http://www.3dluvr.com/content/

11 Vedlegg

Innholdsfortegnelse for vedlegg