Datavetenskap

Magnus Göransson

Ulf Karlsson

Vidareutveckling av ett grafiskt användargränssnitt för duellsimuleringar

---

Examensarbete, C-nivå

2002:28

Vidareutveckling av ett grafiskt användargränssnitt för duellsimuleringar

Magnus Göransson

Ulf Karlsson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Magnus Göransson

Ulf Karlsson

Godkänd, 2002-06-05

Handledare: Thijs Holleboom

Handledare Saab Bofors Dynamics AB:
Darek Sadlakowski

Examinator: Tim Heyer

Sammanfattning

Två applikationer, DuellSim och VisualSim, är resultatet av två tidigare examensarbeten på Saab Bofors Dynamics AB. DuellSim är en applikation för att simulera dueller mellan luftmål och ett luftvärnsrobotsystem. VisualSim är en applikation som visualiserar de scenarion som simulerats.

En önskan från Saab Bofors Dynamics AB var att integrera dessa två applikationer så att en simuleringscykel kunde köras från VisualSim. Detta innebär att ett scenario skulle kunna definieras, simuleras och visualiseras direkt från VisualSims grafiska gränssnitt. Denna önskan formade grunden för detta examensarbete.

Arbetet delades upp i två delar. Först utfördes ett analysarbete som innefattade framtagning av kravspecifikation och designdokument. Sedan utfördes implementation och tester. Detta dokument är en rapport för detta utvecklingsarbete.

Further development of a graphical user interface for duel simulations

Abstract

Two applications, DuellSim and VisualSim, are the result of two earlier student projects at Saab Bofors Dynamics AB. DuellSim is an application for simulating duels between aerial targets and a ground based air defense system. VisualSim is an application for visualizing the scenarios that has been simulated.

A request from Saab Bofors Dynamics AB was to integrate these two applications and make it possible to run an entire simulation cycle from VisualSim. This means that a scenario should be able to be defined, simulated and visualized directly from VisualSim's graphical user interface. This request formed the basis for this bachelor's project.

The work was divided into two parts. The first part consisted of analysis of the task, with creation of a requirement specification and design documents. The other part consisted of implementation and tests. This document is a report of the development.

Förord

Vi vill tacka:

- Darek Sadlakowski – handledare Saab Bofors Dynamics AB
- Thijs Holleboom – handledare Karlstads universitet
- Trolltech support – för att snabbt givit hjälp när problem uppstått.
- Övriga anställda på Saab Bofors Dynamics AB – för trevligt bemötande.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

I detta kapitel kommer vi översiktligt att gå igenom hur rapporten är uppbyggd och vilka delar som ingår i de olika kapitlen m m.

1.1 Allmän information

Detta examensarbete är på 10 poäng och har gått på halvfart under hela vårterminen 2002. Examensarbetet är en del av dataingenjörsprogrammet vid Karlstads universitet, och är nödvändigt för att få ut en examen i slutet av utbildningen. Examensarbetet har utförts på Saab Bofors Dynamics AB (härefter refererat till som SBD) i Karlskoga.

1.2 Översikt

Förutom detta inledande kapitel så innehåller rapporten åtta kapitel. I kapitel 2 ger vi en bakgrund och beskrivning av de tidigare arbeten som lett fram till detta examensarbete. Nästa kapitel handlar om examensarbetets själva uppgift och syfte. Kapitel 4 handlar om arbetsmetoden, där vi steg för steg beskriver de metoder som användes för att arbeta fram kravspecifikationen, designen m m. I kapitel 4 berättar vi även lite om de hjälpmedel och utvecklingsverktyg som vi använt under arbetets gång. I kapitel 5 diskuterar vi hur själva arbetet fortlöpt, gentemot beskriven metod, samt de resultat vi nått. I kapitel 6 tar vi upp de problem som uppstått och eventuella lösningar på problemen. Kapitel 7 och 8 tar upp erfarenheter samt rekommendationer, och i kapitel 9 avslutar vi rapporten med en slutsats av arbetet.

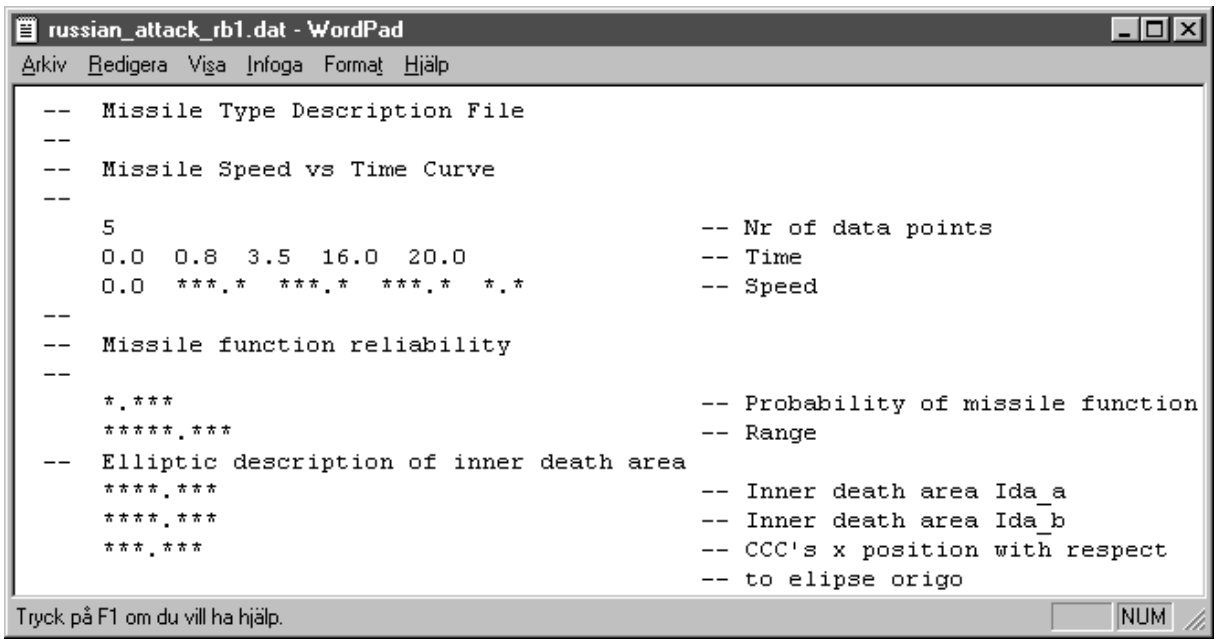
Bilaga A innehåller den kravspecifikation vars framtagning beskrivs i kapitel 5. Bilaga B innehåller ett klassdiagram som beskriver applikationens uppbyggnad. Bilaga C innehåller sekvensdiagram som beskriver olika specifika händelser i applikationen. Bilaga D och E innehåller olika typer av tester på applikationen och resultatet av dessa tester. Bilaga F innehåller skärmdumpar från den vidareutvecklade applikationen.

2 Bakgrund

Två tidigare examensarbeten samt en sommarpraktik vid SBD har lett fram till programmen DuellSim, [1], och VisualSim, [2] och [3]. Detta är två helt olika applikationer som i nuläget körs oberoende av varandra. Trots detta har de en viss koppling, som beskrivs ytterligare nedan. Ett önskemål från SBD har varit att integrera programmen så att DuellSim kan köras från VisualSims grafiska gränssnitt.

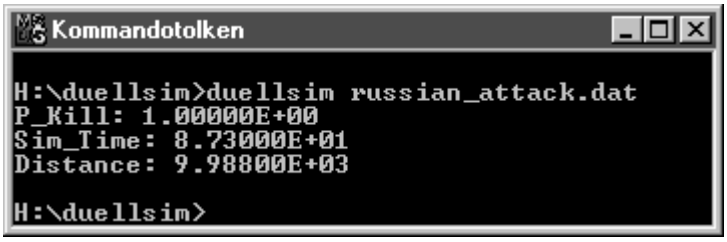
2.1 Beskrivning av DuellSim

DuellSim är ett simuleringsprogram för duellsimuleringar mellan ett luftvärnsrobotsystem och luftmål. Det man vill simulera är luftvärnsrobotsystemets flermålskapacitet, d v s kapaciteten att kunna bekämpa flera mål under kort tid. Varje simulering beskriver ett scenario med aktörer som modellerats för att på bästa sätt återskapa verkligheten. De aktörer som modellerats är skyddsobjekt, stridsledning, eldledningssensor, spaningssensor, robotar och mål. Mål kan vara av typen attackflygplan, kryssningsmissiler eller helikoptrar. De olika aktörerna i scenariot får sin data för egenskaper och prestanda genom att läsa in dem från textfiler (s k datfiler) som skapats manuellt av användaren, se Figur 2.1 (värden som av sekretesskäl ej kan visas har ersatts med asterisktecken).



Figur 2.1: En textfil som används som indata till DuellSim, en s k Datfil

Datfilerna är uppbyggda i en hierarkisk trädstruktur (se Figur 5.1), och DuellSim startas från kommandotolken (dosprompten) med en "rot"-datfil som argument (se Figur 2.2).



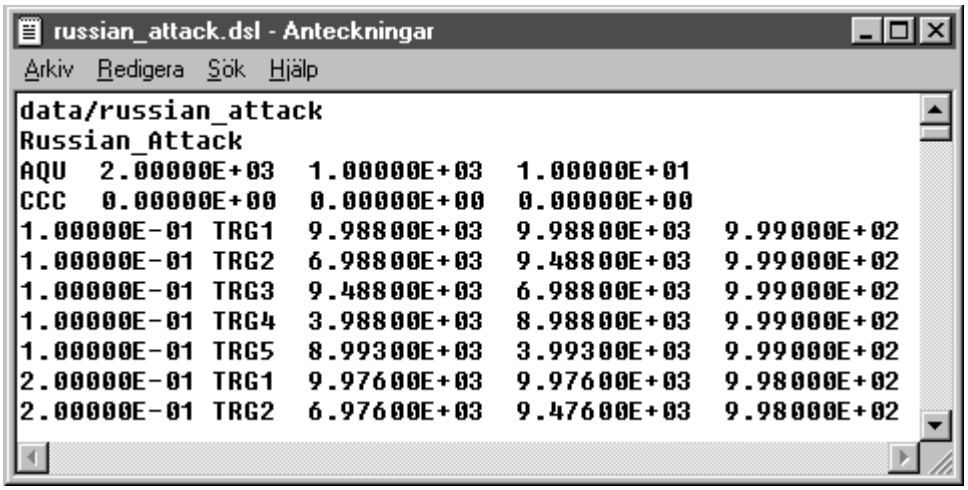
Figur 2.2: Resultatet av en exekvering av DuellSim

Simuleringen är av typen Monte Carlo, d v s ett scenario simuleras vanligtvis ett par tusen gånger, och sedan väljs en av simuleringarna slumpvis ut. Vad som skiljer dessa åt är faktorer som slumpas fram i programmet, som t ex sannolikheten att ett mål upptäcks. Målens färdriktning och liknande är dock alltid samma i alla simuleringar. Under exekveringen ger DuellSim som utdata en loggfil som beskriver händelserna för den utvalda simuleringen. I kommandotolken skrivs även sannolikheten ut för att alla mål bekämpats (i Figur 2.2 är sannolikheten 100%), samt den totala simuleringstiden (87,3 sekunder) och distansen till det mål som upptäckts längst bort (9988 meter) för scenariot. Dessa värden är medelvärden för Monte Carlo-simuleringen. Denna information var av större betydelse innan VisualSim fanns, eftersom det var på detta sätt man då fick information om själva simuleringen.

2.2 Beskrivning av VisualSim

VisualSim är ett visualiseringsprogram för duellsimuleringar genererade av DuellSim. Programmet ger användaren möjlighet att visuellt verifiera simuleringen från det scenario man skapat med DuellSim. Som man kanske kan ana av namnet så har VisualSim ett grafiskt användargränssnitt och är därför något mer användarvänligt än DuellSim. VisualSim kan ses som en spelare för simuleringar, dels med funktioner som play, stop, spola fram och tillbaka o s v, och dels med en stor rityta som visar själva scenariosimuleringen med dess aktörer.

Som indata till VisualSim används de loggfiler som skapats av DuellSim. Loggfilerna innehåller de koordinater där spaningssensorn och stridsledningen ska placeras. Dessa objekt behåller sina positioner under hela simuleringen, och dess koordinater är i förhållande till skyddsobjektet som alltid är placerad i origo på ritytan. Sedan följer en lång lista med nya positioner för målen och robotarna för varje nytt tidssteg. Varje rad beskriver aktuell tid, objektets namn, x-koordinat, y-koordinat och z-koordinat.



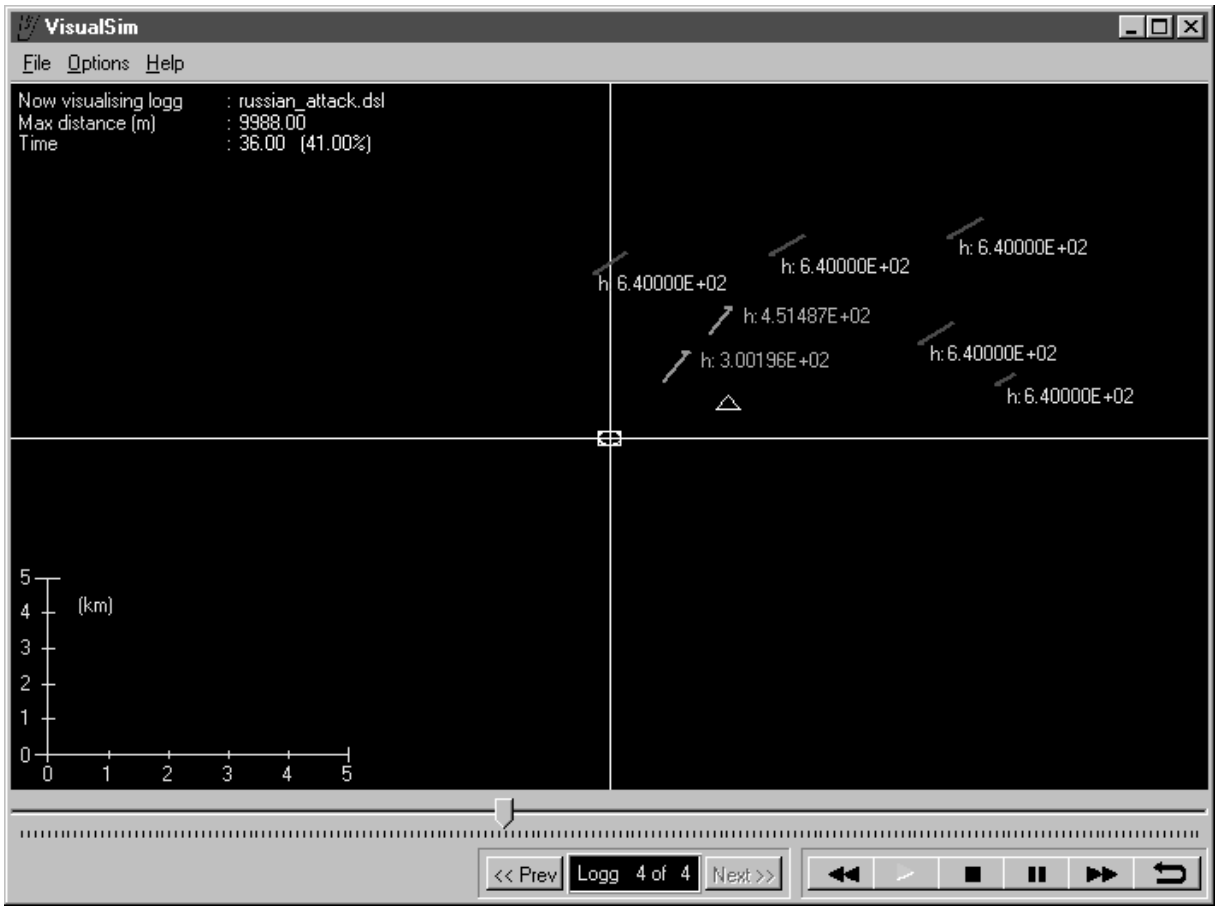
Figur 2.3: Exempel på en loggfil genererad av DuellSim

I Figur 2.3 kan vi se ett scenario innehållande fem stycken mål som kommer att dyka upp i övre högra hörnet på ritytan (x- och y-koordinaterna är positiva). Tidssteget i det här scenariot är 0.1 sekunder, och man kan se om man jämför objektet TRG1s position vid 0.1 sekunder med positionen vid 0.2 sekunder att objektet har ändrat position. Robotarna finns inte med så här tidigt i loggfilen eftersom spaningssensorn ännu inte hunnit upptäcka några mål.
För varje tidssteg uppdateras VisualSims rityta, där de nya positionerna för robotarna och målen ritas ut. Ritytans uppdateringar görs i korrekta tidssteg i förhållande till tidsstegen i loggfilerna, detta innebär att VisualSim presenterar simuleringarna grafiskt i realtid, förutsatt att vissa prestandakrav på hårdvara uppfylls.

Sammanfattningsvis kan man alltså säga att simuleringen sker i simuleringskärnan DuellSim, och resultatet spelas upp i visualiseringsprogrammet VisualSim.

2.2.1 Beskrivning av de olika aktörerna

Figur 2.4 visar VisualSim under exekvering, med de aktörer som ingår i ett scenario. Bilden visar ett scenario kallat ”Russian attack” där fienden försöker slå ut skyddsobjektet i origo medan luftvärnssystemet försöker bekämpa dessa. Aktörerna beskrivs ytterligare nedan.



Figur 2.4: Visualisering av ett scenario där alla aktörerna förekommer

- *PO* - Protected Object (Skyddsobjekt) är som namnet antyder det objekt på ritytan som ska skyddas och som målen ska försöka slå ut. Skyddsobjektet utgör en referenspunkt, d v s ett globalt origo för de övriga aktörerna i scenariot. Skyddsobjektet ritas som en vit cirkel i VisualSim (denna syns dock dåligt i figur 2.4 eftersom även CCC ligger i origo).
- *CCC* - Command Control Center (Stridsledning) är det objekt som gör de taktiska besluten, d v s bestämmer vilka mål som ska bekämpas. Stridsledningen placeras oftast i närheten av skyddsobjektet och ritas som en vit fyrkant i VisualSim.
- *AQU* - Aquisition Unit (Spaningssensor) är det objekt som har till uppgift att spana efter mål och rapportera till Stridsledningen. Spaningssensorn ritas som en vit triangel i VisualSim.
- *FCU* - Fire Control Unit (Eldledningssensor) sitter ihop med Stridsledningen och ritas därför inte ut i VisualSim. Eldledningssensorn är den enhet som innehåller och avfyrar de robotar som skall bekämpa målen.
- *Mål* - Som vi nämnt i kapitel 2.1 kan det förekomma tre olika typer av mål i ett scenario: attackflygplan, kryssningsmissiler och helikoptrar. Målen ritas som röda pilar i

VisualSim, och längden på pilen anger målets hastighet. Vad målet är av för typ går inte att urskilja förutom att om det är en helikopter så ritas ett litet "h" ut bredvid pilen. Målen förflyttar sig vid varje tidssteg enligt koordinaterna i loggfilen.

- *Robot* - Robotarna ritas som gröna pilar i VisualSim, och som för målen är pilens längd ett mått på robotarnas hastighet. Robotarna förflyttar sig vid varje tidssteg enligt koordinaterna i loggfilen.

3 Uppgift

Uppgiften bestod av att vidareutveckla visualiseringsprogrammet VisualSim så att en simuleringscykel skulle kunna köras helt och hållet från det grafiska användargränssnittet.

Simuleringscykeln skulle se ut på följande sätt: Användaren skapar först ett scenario genom att mata in olika värden i det grafiska användargränssnittet, motsvarande de värden som tidigare skrivits direkt i datfilerna. VisualSim skapar sedan automatiskt de datfiler som behövs för scenariot. Därefter anropas programmet DuellSim som genererar en loggfil. Denna loggfil laddas sedan in i VisualSims spellista.

I uppgiften ingick även att i mån av tid göra om presentationen i 2D till 3D.

3.1 Syfte

Syftet med vidareutvecklingen var att öka användarvänligheten och göra det lättare för användaren att skapa, kontrollera, ändra och presentera scenarion, utan att behöva skifta mellan två olika program.

Som vi nämnt i kapitel 2.1 så skapades DuellSims datfiler manuellt av användaren med hjälp av en vanlig enkel texteditor, och detta kunde resultera i att man fick oönskade fel i datfilerna eftersom texteditorer varken känner till formatet på dessa filer, eller giltiga värden för datan till aktörerna. Datfilerna gav heller inte speciellt mycket information för den nye användaren, det kunde vara ganska svårt att tolka och förstå alla olika värden i filerna.

Genom att låta VisualSim ta hand om inmatningen och presentation av värden för datfilerna så skulle man även, genom koll av gränsvärden, kunna förhindra fel uppkomna p g a av den mänskliga faktorn, såsom att skriva fel värden eller tecken för en variabel, t ex ett negativt värde på en sträcka, samt öka användarens förståelse för värdena.

4 Metod

För att lösa uppgiften har vi grovt följt den utvecklingsmetod som används på SBD och på de flesta andra företag. Vi började med inläsning för att förstå uppgiften bättre. Efter detta utformades en kravspecifikation där alla krav på mjukvaran definierades. Detta dokument användes sedan vid utformningen av designen.

Kravspecifikationen och designdokumenten användes vid implementationen. Slutligen gjordes tester på programmet för att säkerställa programmets funktionalitet och korrekthet.

I detta kapitel gås endast metoder och tekniker igenom, själva arbetet och resultat beskrivs i kapitel 5.

4.1 Tidsplan

	v.5	v.6	v.7	v.8	v.9	v.10	v.11	v.12	v.13	v.14	v.15	v.16	v.17	v.18	v.19	v.20	v.21	v.22
Rapportskrivning		10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	100%	10%	50%
Inläsning	100%	40%																
Kravspecifikation		50%	90%	40%														
Granskning					5%	5%												
Design				50%	85%													
Implementation						85%	90%	90%	80%	80%	70%	70%	60%	70%	30%		10%	
Test									10%	10%	20%	20%	30%	20%	60%		80%	
Presentation																		50%
Summa:	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Tabell 4.1: Beräknad tidsplan för utförandet av examensarbetet

Innan vi satte igång med själva arbetsuppgiften så behövde vi en tidsplan att följa. Tillsammans med handledaren på SBD kom vi överens om att vi skulle använda den utvecklingsmetod som används på företaget. Genom att följa de moment som ingår skulle vi få en känsla för hur det kan gå till ute i arbetslivet. Det var också vår handledare på SBD som tog fram tidsplanen i Tabell 4.1 eftersom det var han som hade mest erfarenhet och känsla för ungefär hur lång tid man behöver i varje arbetssteg.

4.2 Inläsning

Inläsning har till syfte att inhämta de kunskaper som behövs för att utföra arbetet. Detta kan innebära att man kan behöva lära sig ett nytt programmeringsspråk, lära sig att använda något nytt utvecklingsverktyg, läsa igenom tidigare dokument (ifall ens arbete går ut på att vidareutveckla), o s v. Denna fas är viktig för att lägga en god kunskapsgrund och för att kunna utföra ett så bra jobb som möjligt.

4.3 Kravspecifikation

Denna fas handlar om att ta fram en kravspecifikation. Kravspecifikationen är ett dokument innehållande de krav som ställs på programvaran som skall tas fram. Dessa krav är sådana som måste uppfyllas för att programmet skall anses korrekt och kan vara både krav på funktionalitet såväl som på utseende.

Kravspecifikationen innehåller även såkallade börkrav, d v s krav som skall uppfyllas i mån av tid men som inte är nödvändiga för att programmet skall bli godkänt. Kravspecifikationen kan ses som ett kontrakt mellan oss och SBD.

4.4 Design

Designfasen är den kreativa processen att hitta en lösning till ett problem. I designen formas systemet så att det uppfyller alla krav i kravspecifikationen.

Anledningen till att man gör en design är för att underlätta implementationen genom att skapa en modell där problemet har delats upp i mindre och tydligt avgränsade bitar. Ett vanligt förekommande modelleringsspråk är UML (Unified Modeling Language) [11], ett visuellt modelleringsspråk för objektorienterad analys och design.

4.5 Implementering

I implementationsfasen realiseras den modell som tagits fram i designfasen. Rent konkret handlar denna fas om att programmera och få fram den önskade produkten. Denna fas tillsammans med designfasen kan repeteras ett antal gånger eftersom det kan hända att man hittar problem som inte går att lösa med den aktuella designen. Det kan då vara nödvändigt att ändra designen allteftersom dessa problem dyker upp.

4.6 Granskning

Efter varje fas i utvecklingsarbete granskas de dokument som producerats innan nästa fas kan påbörjas. D v s innan designfasen kan påbörjas skall kravspecifikationen godkännas och innan implementationen kan börja skall designen godkännas o s v. Granskningarna syftar till att hitta fel eller problem. Granskarna kommer med kommentarer och förslag på ändringar och efter att dessa uppfyllts godkänns fasen.

4.7 Testning

Testfasen har till syfte att kontrollera att produkten uppfyller kravspecifikationen. Dessa tester kan vara av olika karaktär, förutsättningar och storlek. De kan dessutom ske i olika faser av utvecklingen av programvaran, parallellt med eller efter implementationen. De olika testen kan, beroende på karaktär, utföras av antingen programmeraren eller speciellt utsedda testare.

4.8 Utvecklingsverktyg och hjälpmedel

För att underlätta vårt arbete, dels med framtagande av olika dokument, dels med implementation, har vi använt oss av ett antal olika verktyg, dessa beskrivs i detta kapitel.

4.8.1 Rational Rose

Rose [5], från företaget Rational, är ett program som används för att hjälpa till vid utformning och design av system. Det är ett väl beprövat verktyg som idag används i många större företag.

Rational Rose är ett grafiskt verktyg som använder UML för att skapa bl a klassdiagram, sekvensdiagram och Use Cases. Dessa används för att få en modell över hur systemet kommer att se ut. Man kan definiera och deklarera alla de klasser som hör till systemet, och man har en bra kontroll på klassers relationer, funktioner och attribut. Rose kan utifrån skapade diagram generera kodskelett, s k stubbar, i ett flertal olika programspråk.

4.8.2 Microsoft Visual C++

Visual C++ [6] från företaget Microsoft är idag, enligt dom själva, antagligen det mest använda verktyget för att skapa applikationer för Windows i programspråket C++. Microsoft Visual C++ är ett mycket kraftfullt utvecklingsverktyg som innehåller en mängd finesser och komponenter för att underlätta arbetet för programmeraren. Bland dessa kan nämnas den avancerade debuggern. Med hjälp av denna kan man hitta orsaken till olika sorters fel som uppstått vid exekvering, t ex krascher, och lätt lösa dem.

4.8.3 Qt 3.0

Qt [4] är ett C++ klassbibliotek skrivet och distribuerat av det norska företaget Trolltech. Qt tillhandahåller funktionalitet främst för att bygga plattformsoberoende användargränssnitt i C++, men kan även användas till andra områden där det krävs plattformsoberoende. Qt fungerar bland annat på plattformarna MS/Windows, Unix/X11 och Macintosh. Det är helt

och hållet objektorienterat, vilket innebär att det är enkelt att bygga ut och att det tillåter programmeraren att programmera "modulärt".

Qt släpptes i sin första version 1996 och har sedan dess använts i tusentals lyckade applikationer världen över, bl a är Qt basen i den populära fönsterhanteraren KDE för Linux.

Vid installationen av Qt 3.0 medföljer också ett grafiskt utvecklingsverktyg, kallat Qt Designer, för att skapa t ex meddelanderutor, mainwindows och wizards (från och med nu används namnet *dialogruta* som ett samlat begrepp på dessa). Här bestämmer man enkelt utseendet på sin dialogruta genom att helt enkelt dra olika objekt i form av t ex tryckknappar och textfält in på arbetsytan. I Qt Designer kan man också direkt ange egenskaper för sina objekt, t ex defaultvärden, storlek, färg och utseende. Man kan även koppla olika signaler till slots, d v s man anger vilken/vilka funktioner som skall anropas när en användare klickar på en viss knapp.

När man sedan är nöjd med sin dialogruta kan man från designerfilerna generera basklasser i programspråket C++. Man skapar sedan egna klasser i sitt C++-projekt, och låter dessa ärva från basklasserna. Genom att använda sig av arv kan man lätt ge klasserna funktionalitet utan att ändra i basklasserna. Man kan även gå tillbaka till Qt Designer och ändra utseende på sin dialogruta utan att förstöra funktionaliteten. En annan fördel med Qt Designer är att man inte behöver gå in i varje klass och lära sig alla dess metoder för att veta hur man ska sätta olika värden, allt sådant sker grafiskt.

5 Resultat

I det här kapitlet beskrivs resultatet av vårt arbete där vi följt de metoder som beskrivs i kapitel 4. Först beskrivs inläsning av tidigare material, efter detta beskrivs vårt arbete och resultat med att ta fram en kravspecifikation för applikationen. Sedan följer designfasen där vi redovisar vårt arbete med att ta fram klassdiagram och sekvensdiagram, vi beskriver även kort de klasser som införts. Implementationen beskrivs kort och efter detta följer ett avsnitt som beskriver olika granskningar som gjorts på vårt arbete. Mot slutet av kapitlet beskrivs det arbete som gjorts med att testa applikationen. Vi avslutar kapitlet med att jämföra tidsåtgången för de olika delarna i arbetet med den planerade tidsplanen.

5.1 Inläsning

Eftersom detta arbete bygger på tidigare arbeten krävdes en hel del genomgång av gamla dokument innan vi kunde börja med vår egen del av arbetet. Detta innebar att vi fick läsa, analysera och sätta oss in i de arbeten som gjorts tidigare och som är förlagorna till vårt examensarbete, se [1], [2] och [3]. Förutom själva rapporterna behövde vi även gå igenom kravspecifikationer, designdokumet (exempelvis klassdiagram och sekvensdiagram) och källkod för att få en bra inblick hur programmen är uppbyggda. Eftersom vi skulle vidareutveckla ett redan befintligt program så var det viktigt att vi förstod hur allt fungerade och hängde ihop för att kunna göra ett så bra resultat som möjligt.

5.2 Framtagande av kravspecifikation

För att vara säkra på att få till bra krav som kändes korrekta och otvetydiga hade vi fyra olika tillvägagångssätt:

- **Granskning av exjobbsspecifikationen** – Specifikationen som beskrev arbetsuppgiften lästes igenom ett flertal gånger och utifrån denna skapade vi krav. Dessa krav var inte speciellt specifika men gav oss en rätt så bra idé om den önskade funktionaliteten.
- **Utfrågning av uppdragsgivare på SBD** – Genom att diskutera de krav vi kommit fram till med vår handledare på Bofors fick vi hjälp och vägledning både när det gällde formulering av kraven och med att hitta fler.

- **Användarsituationer** – Genom att tänka igenom vilka situationer som skulle kunna uppstå vid användandet av programmet kunde vi bestämma krav för dessa situationer.
- **Prototyper** – Genom att snabbt skapa prototyper av dialoger i Qt Desiger kunde vi komma fram till krav specifika för just den dialogen.

Sedan var det bara att försöka koppla ihop alla delar. Kravspecifikationen hölls på en ganska abstrakt nivå och gick inte in för mycket i detalj. Dels för att underlätta för oss själva, vi skulle bli väldigt bundna i designfasen om kravspecifikationen var allt för detaljerad och dels för att inte allt för lång tid skulle gå åt till att utforma kravspecifikationen. Kontakten med beställaren behöll vi under hela utformningen av kravspecifikationen. När vi ansåg att den var klar lämnades den in till handledaren för kommentarer. Handledaren som tyckte att vi hade fått till en bra kravspecifikation hade endast ett par saker att påpeka. Efter att ha åtgärdat dessa, vilket innebar att redigera ett par punkter och lägga till något krav, blev kravspecifikationen till slut godkänd och låst, och designfasen kunde starta. Kravspecifikationen hittas i bilaga A.

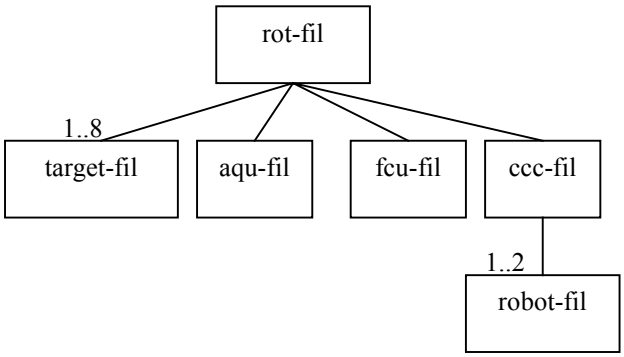
5.3 Framtagande av designen

Som nämnts i kapitel 2 så bygger vårt arbete på tidigare arbeten. Detta innebar att det sedan tidigare fanns ett klassdiagram över VisualSim. Detta var dock inte komplett p g a att applikationen hade fått en del förbättringar och nya finesser. En hel del metoder och variabler hade lagts till i källkoden, metoder och variabler som inte fanns med i det befintliga klassdiagrammet. Vi var tvungna att komplettera detta klassdiagram så att det stämde med den nuvarande versionen av VisualSim innan vi kunde börja med vår egen design. När detta väl var gjort kunde vi koncentrera oss på den huvudsakliga uppgiften. Det var dags att börja fundera på hur man skulle få VisualSim och DuellSim att fungera tillsammans, d v s få den önskade simuleringscykeln att fungera, på ett sätt som uppfyllde kraven i vår kravspecifikation.

5.3.1 Introduktion

DuellSim skapar en loggfil som beskriver resultatet av ett scenario. Informationen som behövs för att simulera ett scenario lagras i textfiler uppbyggda i en trädstruktur (se Figur 5.1). Rotfilen är den som används som argument vid exekvering av DuellSim, och den har i

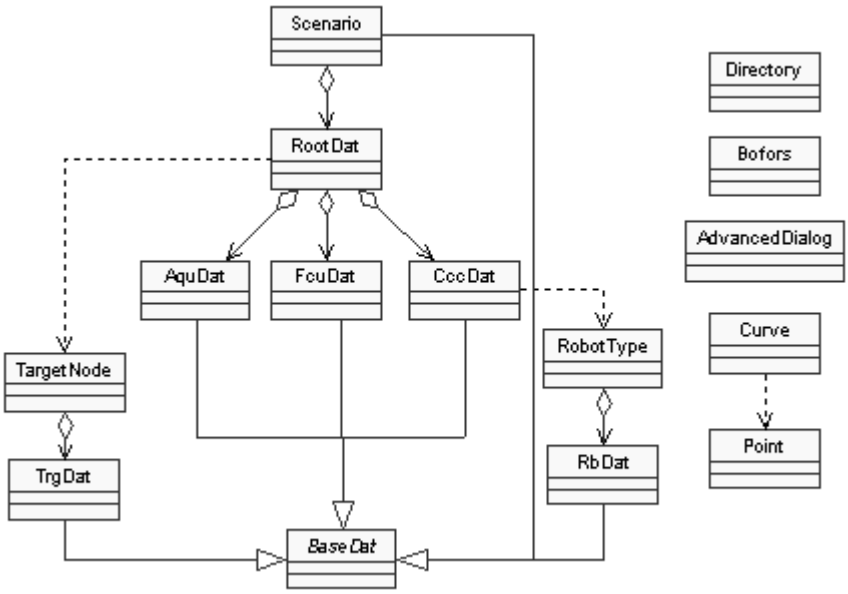
sin tur information om vilka ytterligare filer som används vid scenariot. Dessa filer ska i den nya versionen av VisualSim kunna skapas utifrån information som användaren matar in.



Figur 5.1: Länkningen av de olika textfilerna (datfilerna)

5.3.2 Klassöversikt

Eftersom informationen lagras i separata filer som är ordnade i en viss hierarki kändes det naturligt att skapa nya klasser innehållande respektive fils data och som var ordnade på samma sätt som filerna. Enligt kravspecifikationens krav 3.2.1.a skulle även ett nytt filformat införas, en scenariofil, som håller reda på vilken rotfil som används m m. Även för denna fil skapades en ny klass. Förutom dessa klasser skapades även några klasser för att hantera bl a punkter och kurvor eftersom dessa är vanligt förekommande i filerna. Figur 5.2 visar ett förenklat klassdiagram. Klasserna i denna figur beskrivs nedan. Se bilaga B för ett mer detaljerat klassdiagram. Klassdiagrammet skapades i utvecklingsverktyget Rose som beskrivs i kapitel 4.8.1.



Figur 5.2: Förenklad version av klassdiagrammet i bilaga B

BaseDat En abstrakt klass som innehåller ett par rent virtuella metoder avsedda att läsa in / spara datfiler, samt en samling metoder för enkel läsning och skrivning av data från och till datfiler med deras speciella utseende.

Scenario Denna klass används för att hålla reda på allmän information om ett scenario, t ex filnamn på rot-datfilen, kommentarer m m. Ärver från BaseDat.

RootDat Denna klass blir navet vid editering av scenarion och ansvarar för att all information läses in från de datfiler som ett scenario består av. Klassen lagrar dessutom själv all den information som finns i rot-datfilen. Ärver både från klassen BaseDat samt den huvuddialogruta som genererats från Qt Designer, och är alltså själv en dialogruta. Figurerna i bilaga F.2 och bilaga F.3 visar två olika flikar i denna dialogruta.

RbDat Innehåller den information som finns lagrad / ska lagras i robot-datfilerna (exempel på sådan fil kan ses i Figur 2.1). Informationen beskriver robotens egenskaper som t ex räckvidd och pålitlighet. Ärver från klassen BaseDat.

TrgDat Innehåller den information som finns lagrad / ska lagras i target-datfilerna. Informationen beskriver målets egenskaper som t ex målyta och bekämpningssannolikhet. Ärver från klassen BaseDat.

AquDat Innehåller den information som finns lagrad / ska lagras i aqu-datfilen. Denna information beskriver spaningssensorns egenskaper som t ex radarns rotationshastighet. Ärver från klassen BaseDat.

FcuDat Innehåller den information som finns lagrad / ska lagras i fcu-datfilen. Denna information beskriver eldledningssensorns egenskaper. Ärver från klassen BaseDat.

CccDat Innehåller den information som finns lagrad / ska lagras i ccc-datfilen. Informationen beskriver egenskaperna för stridsledningen, såsom eldhastighet, låstid m m. Ansvarar även för att robot-datfiler blir inlästa. Ärver från BaseDat.

RobotType Innehåller övrig information som rör en robot men som inte lagras i robot-datfilen. Till denna information hör bl a antalet robotar av denna typ, bekämpningssannolikhet m m. Ärver från en av de klasser som genererats från Qt Designer och är alltså, förutom en lagringsklass, en dialogruta (se figur i bilaga F.5).

TargetNode Innehåller övrig information som rör ett mål men som inte lagras i target-datfilen. Till denna information hör bl a startposition, riktningsvektorer m m. Ärver från en av de klasser som genererats från Qt Designer och är alltså, förutom en lagringsklass, en dialogruta (se figur i bilaga F.6).

Curve På ett flertal ställen i datfilerna finns det kurvlistor. Informationen om varje enskild kurva i listan lagras i klassen Curve. Denna innehåller i princip bara en enkel punktlista. Ärver från en av de klasser som genererats från Qt Designer och är alltså en dialogruta (se figur i bilaga F.4).

Point Är helt enkelt bara en klass som lagrar två sammanhängande värden som bildar en punkt. Dessa punkter lagras i punktlistor i bl a klassen Curve.

AdvancedDialog En klass som ärver från en av de Qt genererade klasserna och är alltså en dialogruta (se figur i bilaga F.8). Används för att visa och möjliggöra ändringar av lite mer avancerade egenskaper för stridsledningen. Inmatade värden lagras inte i objekt av denna klass utan dessa lagras i CccDat-objektet.

Directory En dialogruta som används för att sätta sökvägen till programmet DuellSim. Dialogrutan kan ses i bilaga F.7.

Bofors Skapar en Boforslogotyp i 3D, roterande kring sin egen axel. Programmeraren kan bestämma rotationshastighet, vinkel och avstånd.

5.3.3 Sekvensdiagram

För att få bättre överblick över samarbetet mellan olika klasser vid bestämda situationer skapade vi ett antal olika sekvensdiagram. Vi använde oss av verktyget Rose, som finns beskrivet i kapitel 4.8.1. De sekvensdiagram vi skapat är:

- *New/Open scenario* (se bilaga C.1) beskriver händelserna när ett nytt scenario skapas alternativt när ett befintligt scenario öppnas.
- *Save scenario* (se bilaga C.2) beskriver händelserna när ett scenario sparas.
- *Generate log, inga ändringar gjorda innan* (se bilaga C.3) beskriver händelserna när en logg ska genereras och scenariot inte ändrats sedan det sparades senast.
- *Generate log, ändringar gjorda innan* (se bilaga C.4) beskriver händelserna när en logg ska genereras och scenariot har ändrats sedan det sparades senast.

5.4 Implementation

Implementationen påbörjades efter att designen godkänts vid en granskning. Implemenationen skedde i Visual C++, beskrivet i kapitel 4.8.2.

Vi började med att implementera de dialoger som behövdes, se figurer i bilagorna F.2 - F.8. Dessa skapades i Qt Designer. Vi hade ”skal” till dessa från krav- och designfasen som vi

bara gjorde ett par små ändringar på. Från dessa genererades klasser som vi kunde ärva från. Vi kontrollerade sedan att dessa kunde öppnas och stängas på önskat vis.

Efter detta implementerade vi de klasser som lagrar data inläst från datfilerna. Till en början implementerades bara metoderna readFromFile och de metoder som ser till att värdena visas i dialogrutorna, detta för att kontrollera att inläsningen fungerade.

När väl inläsningen fungerade implementerade vi de metoder som anropar DuellSim och tar hand om dess genererade loggfil. Efter detta implementerades metoden writeToFile. Kvar att göra var felkontroller och uppdateringar vid ändring av värden samt diverse småkontroller.

Under arbetets gång upptäckte vi att det behövdes en hel del metoder förutom de som vi tagit upp i designen, dels för att dela upp metoder som blev alldeles för stora och dels för att minska redundansen i koden.

Problem som dök upp under implementationen löstes genom att studera [4], [8], [9] och [10] alternativt genom hjälp från arbetskamrater eller från Trolltechs supportavdelning.

5.4.1 3D och OpenGL

Enligt kravspecifikationens börkrav 3.2.2.3.a - 3.2.2.3.c skulle dagens duellpresentation göras om från 2D till 3D. När vi kände oss klara med alla krav började vi kolla lite på OpenGL och försöka göra en uppskattning om börkravet med 3D skulle hinna uppfyllas. Kunskaper om OpenGL fick vi från [7]. Vi kom fram till att vi antagligen inte skulle hinna färdigt med detta och vi ville inte lämna något halvfärdigt.

Handledaren på Bofors ansåg dock att vi skulle göra något litet i 3D för att testa Qt 3.0s stöd för OpenGL. Vi bestämde oss därför för att göra ett litet roterande B med en pil igenom, Bofors logotyp, och placera denna upp i ena hörnet på VisualSim, se skärmdumpen i bilaga F.1.

5.5 Granskning

5.5.1 Granskning av kravspecifikation

Granskningen av kravspecifikationen skedde av handledaren på SBD som läste igenom den, godkände den och slutligen låste den för vidare ändringar.

5.5.2 Granskning av design

Granskningen av designen gick till som så att vi fick ha en liten presentation för granskarna, som bestod av vår handledare plus ytterligare en anställd på företaget. Vid presentationen

beskrev vi först problemet och arbetsuppgiften, sedan visade vi lösningsförslaget som vi kommit fram till i designfasen. Med hjälp av klassdiagram och sekvensdiagram kunde granskarna se hur vi hade tänkt, och kunde sedan komma med förslag på förbättringar. Designen blev med beröm godkänd och endast ett par saker fanns att påpeka, som t ex att använda oss av Qts färdiga klasser för listor istället för att skapa egna listklasser.

5.5.3 Granskning av källkod

Granskningen av källkoden utfördes av vår handledare på SBD plus ytterligare två anställda på företaget. Granskarna fick ut designdokument och källkod cirka en vecka innan granskningen för att ha tid att gå igenom dessa. Vid granskningstillfället fick vi höra deras åsikter och funderingar angående vår kod i förhållande till designen. Efter granskningen ändrade vi koden enligt de påpekanden vi fått. Detta tog ingen längre tid eftersom inga allvarliga fel i koden hittats.

5.6 Test

Som det framgår av tidsplanen, Tabell 4.1, skedde testning till viss del parallellt med implementationen. Dessa tester var s k White Box tester, d v s man har kännedom om hur koden ser ut och testar specifika fall utifrån dessa kunskaper. Dessa tester utfördes av oss själva. Problem som dök upp vid de parallella testerna åtgärdades på en gång innan vi gick vidare med implementationen.

När vi ansåg oss vara klara med implementationen gjordes en demokörning av applikationen för att kontrollera att alla krav i kravspecifikationen var uppfyllda. Denna kontroll utfördes av vår handledare på SBD. För testresultat vid test gentemot kravspecifikationen, se bilaga D.

Vi gjorde även ett lite större test, ett s k Black Box test. Ett Black Box test innebär att man utformar testfall utan någon kunskap om koden. Testplanen för Black Box testet utformades så att varje testfall innehöll en beskrivning, tillvägagångssätt, förväntat resultat samt faktiskt resultat. Varje testfall i testplanen utformades med tanke att testa systemets funktionalitet vid olika användarsituationer. Denna testplan och testresultat finns i bilaga E.

Ett mer omfattande test, ett test som skulle täcka upp de flesta möjliga händelser, skulle tagit allt för lång tid. Vi ansåg att de tester som gjordes fortlöpande tillsammans med den avslutande testningen räckte gott för att konstatera att programmet fungerade som det var tänkt och uppfyllde kravspecifikationen. De fel som efter dessa tester finns kvar är antagligen

fel som bara dyker upp i situationer som inte kan uppstå såvida man inte tvingar fram dom. Som det framgår av bilaga D och bilaga E har alla tester utförts utan att några fel upptäckts.

5.7 Tidsplan

		v.5	v.6	v.7	v.8	v.9	v.10	v.11	v.12	v.13	v.14	v.15	v.16	v.17	v.18	v.19	v.20	v.21	v.22
Rapportskrivning	*Beräknad*		10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%	100%	10%	50%
	Faktisk				10%	5%							50%	70%	80%	90%	100%	100%	50%
Inläsning	*Beräknad*	100%	40%																
	Faktisk	90%	10%	10%															
Kravspecifikation	*Beräknad*		50%	90%	40%														
	Faktisk	10%	80%																
Granskning	*Beräknad*					5%	5%												
	Faktisk		10%			5%	10%									10%			
Design	*Beräknad*				50%	85%													
	Faktisk			90%	90%	20%													
Implementation	*Beräknad*						85%	90%	90%	80%	80%	70%	70%	60%	70%	30%		10%	
	Faktisk					70%	90%	95%	95%	95%	95%	95%	45%	10%					
Test	*Beräknad*									10%	10%	20%	20%	30%	20%	60%		80%	
	Faktisk							5%	5%	5%	5%	5%	5%	20%	20%				
Presentation	*Beräknad*																		50%
	Faktisk																		50%
Summa:		100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Tabell 5.1: Faktisk tidsplan i förhållande till beräknad tidsplan

Som framgår av Tabell 5.1 låg vi ganska bra till tidsmässigt under hela tiden som projektet pågick. Vi hann klart i god tid med varje fas innan det var dags att börja med nästa, och vi behövde aldrig känna oss stressade. För den skull har vi inte haft dåligt med arbete. Tvärtom har vi haft fullt upp hela tiden men ändå haft kontroll över situationen. Detta borde tyda på god planering och disciplin, samt en väl avgränsad och noga avvägd kravspecifikation. Tyvärr så slarvade vi lite med rapportskrivningen under implementeringsfasen. Förklaringen till detta är enkel, det var så roligt att implementera så rapportskrivningen kom i skymundan.

6 Problem

Detta kapitel tar upp de problem som vi stött på under arbetets gång. Respektive problems karaktär beskrivs liksom lösningen vi kom fram till.

6.1 Uppgradering från Qt 2.0 till Qt 3.0

Den gamla versionen av VisualSim använde sig av Qt 2.0 och eftersom SBD köpt en licens för Qt 3.0 var vi tvungna att göra en portning, d v s byta ut anrop till gamla, nu borttagna, funktioner mot nya, m m. Detta ställde till en hel del problem. Trots att vi följde de portningsanvisningar som fanns på Qts hemsida ville programmet inte kompilera korrekt. Hjälpen vi fick från Qts support hjälpte oss inte heller. Det visade sig till slut vara Visual C++ som inte var korrekt konfigurerat vad gäller länkning o s v.

6.2 Kontroll av gränsvärden i inmatningsrutorna

Enligt kravspecifikationens krav 3.2.2.2.d skall vissa värden ha gränser som de ska hålla sig inom. Enligt krav 3.2.2.2.e skall programmet även säga till när ett värde är felaktigt och sätta det till ett defaultvärde. Dessa två krav vållade oss en hel del problem. Vi löste det dock till slut med hjälp av något som kallas för validatorer, ett Qt specifikt kontrollobjekt som håller reda på gränsvärden och kan utvärdera ett värde med hjälp av dessa gränser. Värden utvärderade med dessa validatorer vara antingen korrekta, ”nästan” korrekta eller felaktiga. Med nästan korrekta menas att värdet eventuellt kan justeras för att bli korrekt. Man måste dock implementera denna justeringsmetod själv, något som vi inte brydde oss om eftersom vi inte ansåg oss behöva denna funktionalitet.

6.3 Implentering av Cancel-funktion

Eftersom det finns möjlighet att ångra inmatningar genom att använda Cancel-knappen måste alltid en kopia av scenariot skapas, åtminstone av den del som har ändrats. Detta ställde till en hel del problem för oss. Dels problemet att hålla reda på orginalobjektet men även att veta om användaren klickar Ok i en nivå ovanför. T ex om användaren väljer att skapa ett nytt mål så måste en kopia av orginallistan med mål då sparas undan (om en sådan inte redan finns) innan dialogrutan kan visas. Klickar användaren sedan på Ok-knappen ska målet sparas undan,

annars skall det tas bort ur listan. Att få reda på vad som hade klickats ställde till en del problem.

6.4 Decimaltal i Spinbox

Att få värdet i spinboxen, se figur 6.1, att öka med steget 0.1 (0.1, 0.2….o s v) verkar vara omöjligt. Detta eftersom man endast ökar värdet med heltal. Det finns dock en liten finess i QSpinboxklassen som kallas prefix. Med prefix menas att man kan definiera vad som ska stå före värdet i spinboxen, och genom att sätta "0." som prefix så fick vi det att se ut som om värdet ökade med en tiondel. Detta såg bra ut i intervallet 0.0 - 0.9. Tyvärr blev det inte rätt när man från 0.9 ökade med 0.1. Man hade ju önskat att man då skulle fått värdet 1.0, men så blev icke fallet. P g a det lilla prefixet fick vi då istället 0.10.

Eftersom vi inte hittade någon lösning på detta problem valde vi istället att använda ett vanligt inmatningsfält.

Figur 6.1: Objektet som av Qt kallas Spinbox

7 Erfarenheter

Tiden på SBD har gett oss en inblick i hur arbetet kan fungera vid framtagandet av ett system, eller som i vårt fall, en vidareutveckling av en applikation. Vi har följt en förenklad version av den utvecklingsmetod som används av företaget men ändå fått nyttiga erfarenheter av både kravanalys och designarbete.

Eftersom vi har arbetat med att vidareutveckla en applikation har vi fått känna på hur det är att sätta sig in i någon annans programkod. En erfarenhet som visat att det är viktigt att kommentera mycket och skriva bra dokumentation, vilket man kanske inte tänkt så mycket på tidigare. Vi har också märkt att det även är lätt att glömma hur man själv tänkt, så för den skull är det också viktigt att fortlöpande dokumentera alla tillägg och ändringar, i såväl design som implementation.

I designfasen använde vi oss, som tidigare nämnts i kapitel 5.3, av utvecklingsverktyget Rational Rose. Vår erfarenhet och uppfattning av programvaran är väl inte direkt till fördel för Rational. Till nackdelarna hör att det är en aning rörigt och inte alltid beter sig som man själv vill. Om man har en stor modell med många klasser och aktörer kan diagrammen bli svåra att överskåda och hantera. Rose kan användas för att generera C++-skelett, men vi har dock enbart använt oss av Rose för att skapa klassdiagram och sekvensdiagram under designfasen.

Övriga utvecklingsverktyg vi använt oss av under projektets arbetsgång var Microsoft Visual C++ och klassbiblioteket Qt från Trolltech. Microsoft Visual C++ är ett verktyg som är mycket enkelt att lära sig och trevligt att använda, samtidigt som det är oerhört kraftfullt. Visserligen har vi inte använt ens hälften av Visual C++s alla funktioner, men det känns ändå som att det är nödvändigt att de finns där för större projekt. Ett och annat problem har dock uppstått under tiden vi arbetat med projekt i Microsoft Visual C++, t ex har vi haft problem när det gäller inställningar i programmet vad avser länkning m m.

Vår personliga uppfattning om Qt är mycket positiv. Tack vare Trolltech har det nu blivit oerhört enkelt att skapa applikationer med grafiskt användargränssnitt. Det grafiska utvecklingsverktyget Qt Designer gör det dessutom ännu enklare att få en överblick över hur ens applikation kommer att se ut för användaren. Qt har också en oerhört väldokumenterad referensmanual för programmeraren, se [4], där man kan hitta i stort sett allt om allt som har med Qt att göra. Om man mot förmodan fortfarande skulle ha frågor om Qt så kan man, om man innehar en licens, vända sig direkt till Trolltechs support med ett mail, och oftast får man

svar efter bara några timmar. Detta har vid ett par tillfällen varit fallet för oss, och kontakten med Trolltechs support har känts väldigt trevlig och personlig. Dom var väldigt duktiga och brydde sig verkligen om att hjälpa oss lösa våra problem.

Trots att vi inte fick så mycket tid över till att jobba med OpenGL har vi ändå fått en liten inblick i hur det fungerar. Det har varit mycket intressant och vi hade gärna sett att vi fått mer tid över till att utveckla presentationen av duellsimuleringen i 3D.

8 Rekommendationer och fortsatt arbete

Vi rekommenderar att någon fortsätter med utvecklingen av VisualSim och gör om den nuvarande presentationen till 3D, dels för att programmet skulle bli bra mycket trevligare att se på, dels för att det antagligen även skulle snabba upp visualiseringen av scenarion, eftersom Qts pixmaps (som används för att rita bilder) tar en oerhörd processorkraft. Därför skulle ett bra grafikkort som stöder OpenGL kunna ta över mycket av denna arbetsbörda.

Om man i fortsättningen använder sig av Qt Designer, låt då egna klasser ärva från de klasser som genererats från Qt Designer istället för att göra tillägg eller ändringar i den genererade koden. Detta förhindrar att gjorda ändringar försvinner vid omgenerering av basklasserna. Om man dessutom slipper bry sig om inställningar som rör dialoger o dyl så får man bara de viktiga delarna kvar. Det gör att den egna koden blir mer lättöverskådlig. Detta är något som inte hade gjorts i den tidigare versionen av VisualSim och detta försvårade förståelsen och vidareutvecklingen till viss del.

9 Slutsats

Vi har skapat en applikation som tillgodoser de önskemål som låg till grund för det här examensarbetet. Applikationen fungerar tillfredsställande och uppfyller alla krav som tagits fram i kravspecifikationen. Efter testfasen kunde vi också konstatera att applikationen känns stabil, och att olika användarfall inte resulterar i oönskade termineringar av programmet.

Valet att inte realisera börkraven 3.2.2.3.a - 3.2.2.3.c var antagligen riktigt. Att göra om allt till 3D hade tagit för lång tid, även om det kanske inte hade blivit så svårt. Vi hade gärna gjort detta arbete, men vi kände att vi inte vill lämna något halvfärdigt. Därför får logotypen räcka tills vidare.

Arbetsuppgiften har varit väldigt kul, och vi har trivts mycket bra här på SBD. Vi har fått bra kontakt med övriga anställda och blivit bra behandlade av allihop här på företaget. Folk har behandlat oss som vilken anställd som helst och det har känts skönt att inte behöva känna sig som en praktikant.

Referenser

[1] Darek Sadlakowski. *Simuleringsprogram för duellsimulering*. Examensarbete, Saab Bofors Dynamics AB, 1999. ISSN 1402-1617

[2] Johan Gustafsson. *Visualiseringsprogram för en duellsimulering*. Examensarbete, Saab Bofors Dynamics AB, 2001.

[3] Magnus Göransson. *VisualSim 1.1*. Sommarpraktik, Saab Bofors Dynamics AB, 2001.

[4] Trolltech, *Qt Reference Documentation*, http://doc.trolltech.com/3.0/index.html, 2002-04-30.

[5] Rational, *Rational Rose v 2001*, http://www.rational.com/products/rose/index.jsp, 2002-04-30

[6] Microsoft, *The Product and Technology Catalog*, http://www.microsoft.com/catalog/, 2002-04-30

[7] Richard S Wright, Jr, Michael Sweet, *OpenGL Super Bible, 2nd edition*, Waite Group Press, 2000, ISBN 1-57169-164-2

[8] Patrick Ward, *Qt Programming for Linux and Windows 2000,* Prentice Hall PTR, 2001, ISBN 0-13-027001-6

[9] Jan Skansholm, *C++ Direkt,* Studentlitteratur, 1996, ISBN 91-44-47931-X

[10] Deitel & Deitel, *C++ How To Program, 2nd edition,* Prentice Hall, 1998, ISBN 0-13-528910-6

[11] OMG, *UML Home Page*, http://www.uml.org, 2002-06-06

A Kravspecifikation

Sammanfattning ***Abstract***

Detta dokument innehåller kravspecifikationen för vidareutvecklingen av VisualSim, ett visualiseringsprogram för duellsimulering.
Den nya versionen, kallad VisualSim 1.2, ska göra det möjligt att köra en hel simuleringscykel där inmatning av data sker grafiskt. Datan lagras i textfiler som körs i programmet DuellSim, se [1], som i sin tur genererar loggfiler som spelas upp i VisualSim 1.2.

1 Omfattning

1.1 Identifikation

Ej tillämpligt

1.2 Systemöversikt

Programmet som detta dokument avser är en vidareutveckling av GUI:et VisualSim 1.1, se [2] och [3], ett visualiseringsprogram för luftduellssimuleringar genererade av programmet DuellSim, se [1].

DuellSim används för att simulera dueller mellan en stationär eldenhet och luftmål som t ex attackflygplan. Simuleringen är av typen Monte Carlo, d v s ett stort antal simuleringar körs för att få hög konfidens. Data angående ett scenario läses in från ett antal textfiler, kallade dat-filer.

Vidareutvecklingen av VisualSim 1.1 ska göra det möjligt att köra en hel simuleringscykel ifrån GUI:et. Simuleringscykeln innebär att de dat-filer som DuellSim behöver för att kunna generera en loggfil för ett scenario skapas med hjälp av VisualSim 1.2. Användaren matar in önskad information i GUI:et. Denna information sparas som dat-filer som sedan används av programmet DuellSim, som i sin tur genererar loggfiler som spelas upp i VisualSim 1.2.

1.3 Dokumentöversikt

Detta dokument specificerar kraven som programvaran VisualSim 1.2 skall uppfylla.

2 Tillämpbara dokument

2.1 Myndighets dokument

Ej tillämpligt

2.2 Övriga dokument

[1] Simuleringsprogram för duellsimulering, Darek Sadlakowski, 1999, ISRN: LTU – EX - - 99/300 - - SE

[2] Visualiseringsprogram för en duellsimulering, Johan Gustafsson, 2001, ISRN: Oru-Te-EXD083-D22/01

[3] VisualSim 1.1 – sommarpraktik av Magnus Göranson 2001

3 Krav

Detta kapitel specificerar de krav som är nödvändiga för att tillförsäkra en lämplig vidareutveckling av programvaran, VisualSim.

3.1 Krav på externa gränsytor

Ej tillämpligt

3.2 Kapabilitetskrav för VisualSim 1.2

3.2.1 Data

Krav:

a) Ett nytt filformat, vsf (VisualSim Scenario File), skall skapas för att hålla reda på vilken dat-fil (rotfilen) och loggfil som används vid ett scenario. Filen skall även innehålla kommentarer om scenariot.

b) Efter att information matats in i dialogrutan som beskrivs i 3.2.2.2, skall informationen sparas i dat-filer för att köras i DuellSim.

c) Alla filer i ett visst scenario skall genom sitt filnamn visa vilket scenario de tillhör, t ex myScenario.dat, myScenario_aqu.dat, myScenario.dsl o s v.

d) En importerad dat-fil, se 3.2.2.2 krav f, skall kopieras och få ett namn passande det scenario det importeras till. Ändringar som görs påverkar inte originalfilen. *Exempel*: *Om filen myScenario_aqu.dat importeras till scenariot yourScenario, skall en ny fil yourScenario_aqu.dat skapas som är en exakt kopia av myScenario_aqu.dat.*

e) Vid import av en dat-fil skall även dat-filer länkade från denna fil importeras och döpas om enligt punkt d.

f) Rotfilen skall ej gå att importera. Istället skall menyalternativet ”spara som” finnas för scenarion, vilket ger samma resultat.

g) Loggfiler skall genereras med hjälp av den information som finns i scenariots nyskapade dat-filer. Genereringen skall ske med DuellSim.

h) En fil innehållande sökvägen till Duellsim skall läsas in vid uppstart. Vid ändring av sökväg skall denna fil uppdateras.

Bör:

a) En loggfil som genererats ska automatiskt laddas in.

b) Vid import av en dat-fil skall endast rätt sorts dat-filer visas i dialogrutan. *Exempel: Man vill importera en AQU dat-fil. Endast de dat-filer som innehåller data för AQU’n kommer att visas.*

3.2.2 Användargränssnitt

3.2.2.1 Meny

Krav:

a) I Arkiv-menyn skall val läggas till för att

- skapa scenario
- öppna scenario
- spara scenario
- spara scenario som
- redigera scenario
- stänga scenario
- generera loggfil
- sätta sökväg till DuellSim

Bör:
a) Valen ovan skall även finnas som snabbvalsknappar under menyn.

3.2.2.2 Dialogruta för inmatning av indata för scenario

Krav:

a) En dialogruta med flikar skall användas för att enkelt kunna hoppa mellan inmatning av de olika aktörernas data. Flikarna skall innehålla fält för inmatning av data för:
   - data som sparas i rotfilen, flik General
   - spaningssensor, flik AQU
   - eldledningsenhet, flik FCU
   - stridsledning inklusive robotar, flik CCC
   - mål, flik Targets

b) All information om objekts egenskaper (såsom positioner, hastigheter), som idag finns lagrade i dat-filerna (d v s indatan), skall kunna matas in manuellt av användaren genom ovan beskrivna dialogruta.

c) För varje värde i dialogrutan skall vid skapandet av ett nytt scenario finnas ett defaultvärde hämtade från ”default”-dat-filer.

d) För vissa data skall användaren endast kunna mata in värden mellan en min och max gräns.

e) Vid inmatning av ogiltigt värde skall programmet meddela detta, och datan ifråga skall sättas till defaultvärde.

f) Vid respektive flik skall användaren kunna importera redan färdiga dat-filer.

3.2.2.3 Presentation av luftduell

Bör:

a) Alla objekt på ritytan i den nuvarande presentationen av luftduellen bör göras om till 3D.

b) 3D-presentationen bör kunna roteras.

c) 3D-presentationen bör vara zoombar.

3.3 Interna gränsytor i CSCI:en

Ej tillämpligt

3.4 Dataelementskrav för VisualSim 1.2

Ej tillämpligt

3.5 Krav på anpassning

Krav:

a) Programmet skall kunna köras på en PC med operativsystemet Windows NT.

Bör:

a) Programmet bör kunna köras på en arbetsstation SUN Solaris.
b) Programmet bör kunna köras på en PC med operativsystemet Linux.

3.6 Dimensionerade krav

Ej tillämpligt

3.7 Säkerhetskrav

Ej tillämpligt

3.8 Sekretesskrav

Ej tillämpligt

3.9 Konstruktionsbegränsningar

Ej tillämpligt

3.10 Kvalitetsfaktorer för programvara

Ej tillämpligt

3.11 Samspel människa – maskin

Krav:

a) Användargränssnittet skall vara användarvänligt.

3.12 Kravens spårbarhet

Ej tillämpligt

3.13 Programmeringsspråk

Krav:

a) Uppbyggnaden av programmet skall vara objektorienterat och programmet skall implementeras i C++ med verktyget Qt 3.0. Koden kompileras med Microsoft Visual C++ 6 kompilatorn.

b) Implementationen skall vara lättförståelig.

c) Koden till programmet skall vara väl kommenterad.

4 Kvalificeringskrav

Detta kapitel skall specificera kvalificeringsmetoder och övriga speciella kvalificeringskrav som är nödvändiga för att tillse att VisualSim 1.2 möter kraven i kapitel 3 och 5.

4.1 Kvalificeringsmetoder

Skall specificera de kvalificeringsmetoder som skall användas för att försäkra att kraven i kapitel 3 och 5 har blivit tillgodosedda. Kvalificeringsmetoder innefattar:

a) **Visning (Demonstration).** Exekvering av VisualSim 1.2 (eller del av VisualSim 1.2) som grundar sig på observerbar funktionell användning och som inte erfodrar en raffinerad instrumentering eller provutrustning.

b) **Analys (Analysis).** Bearbetning av ackumulerad information som erhållits från andra provmetoder. Exempel är tolkning eller extrapolering av provdata.

c) **Kontroll (Inspection).** Den visuella granskningen av programvarukod, dokumentation etc.

4.2 Speciella kvalificeringskrav

Ej tillämpligt

5 Leveransklargöring

Ej tillämpligt

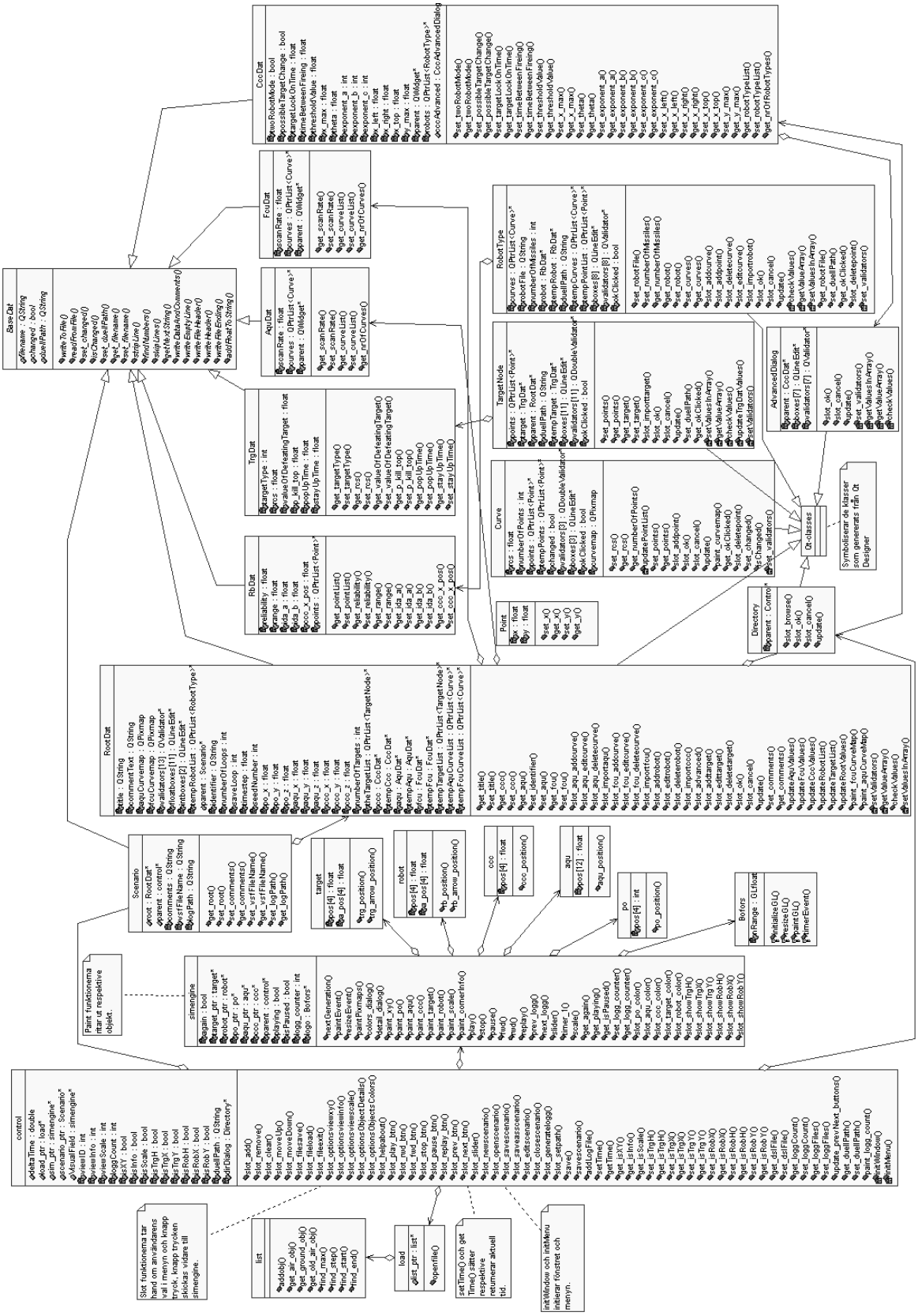
6 Noteringar

Detta avsnitt innehåller allmän information för att underlätta förståelsen av detta dokument.

6.1 Förteckning över använda förkortningar

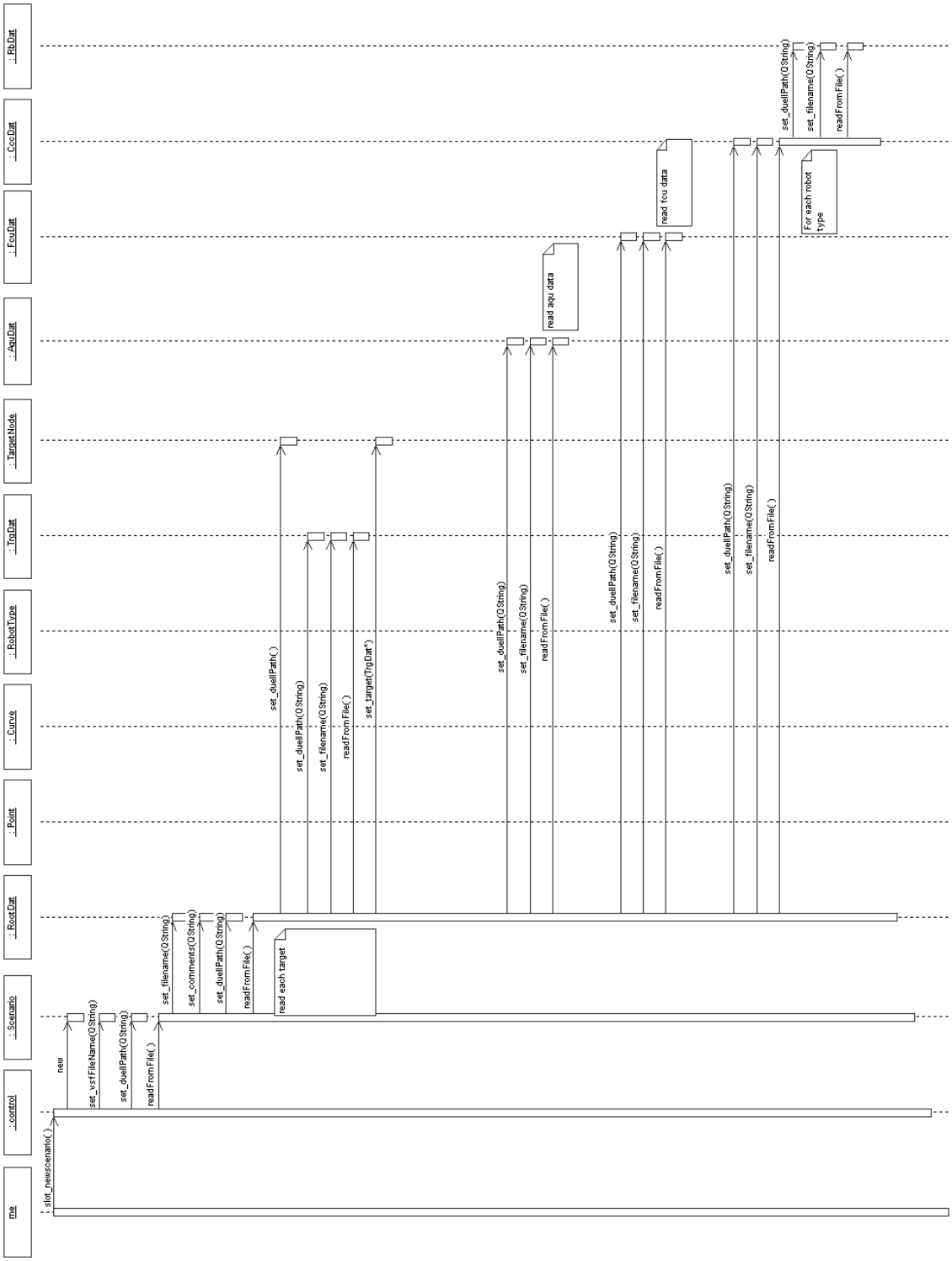
DuellSim	Simuleringsprogram för duellsimulering
VisualSim	Visualiseringsprogram för duellsimulering
AQU	Acquisition Unit
FCU	Fire Control Unit
CCC	Command Control Center

B Klassdiagram

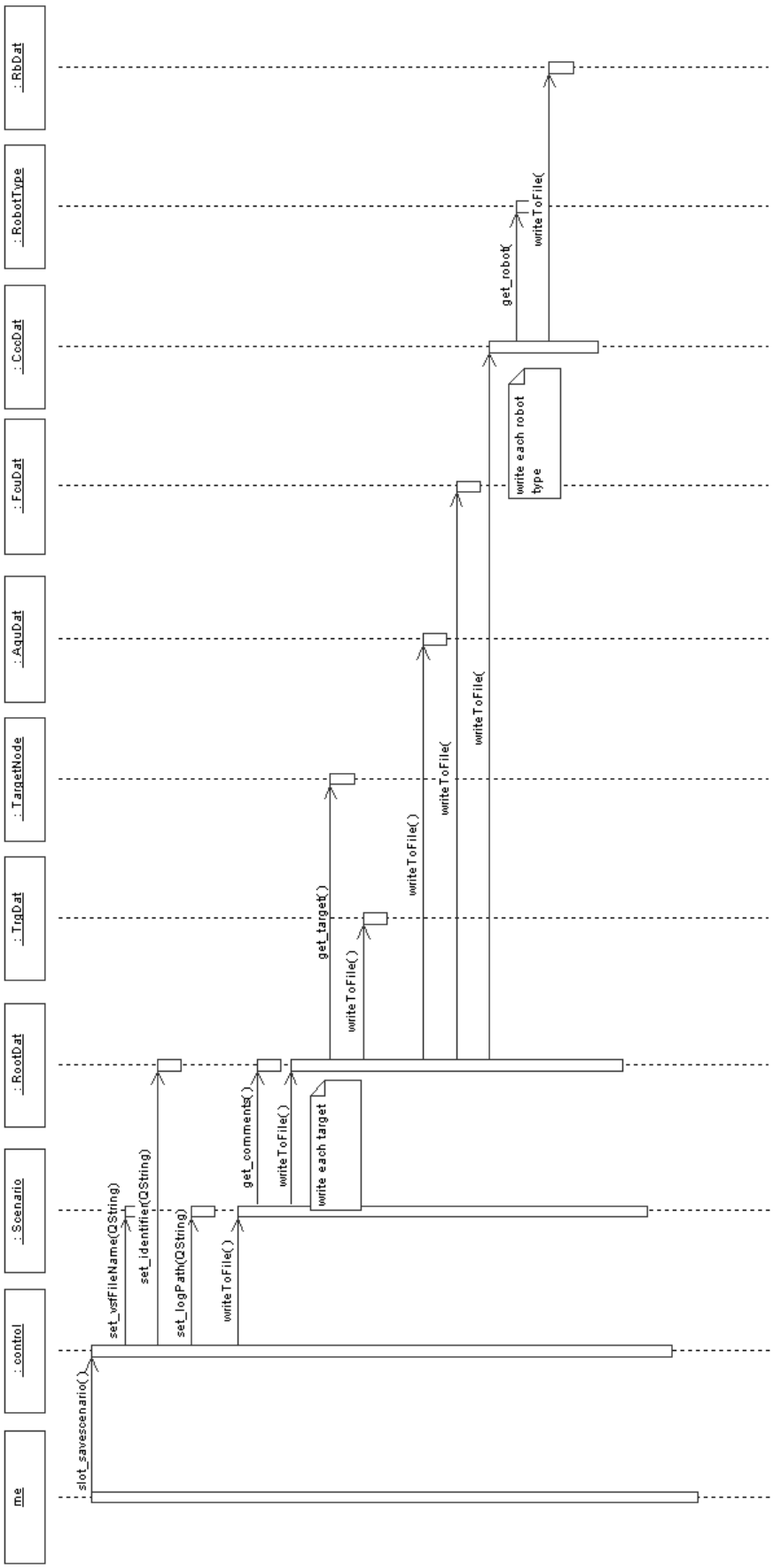


C Sekvensdiagram

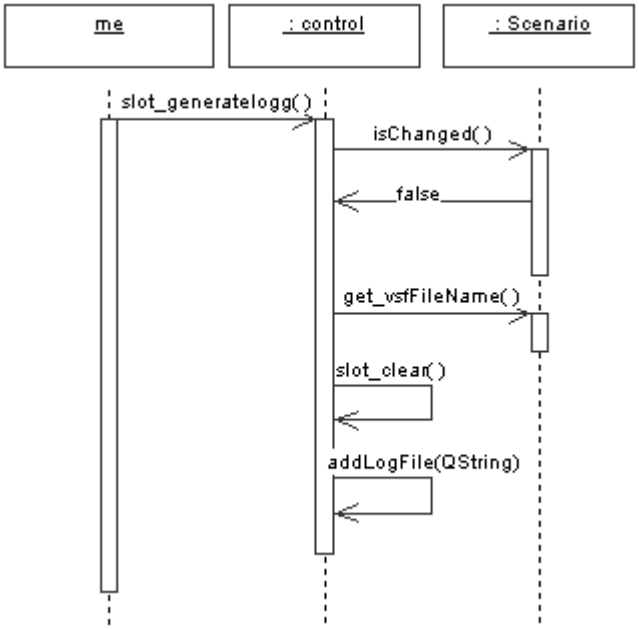
C.1 New/Open Scenario



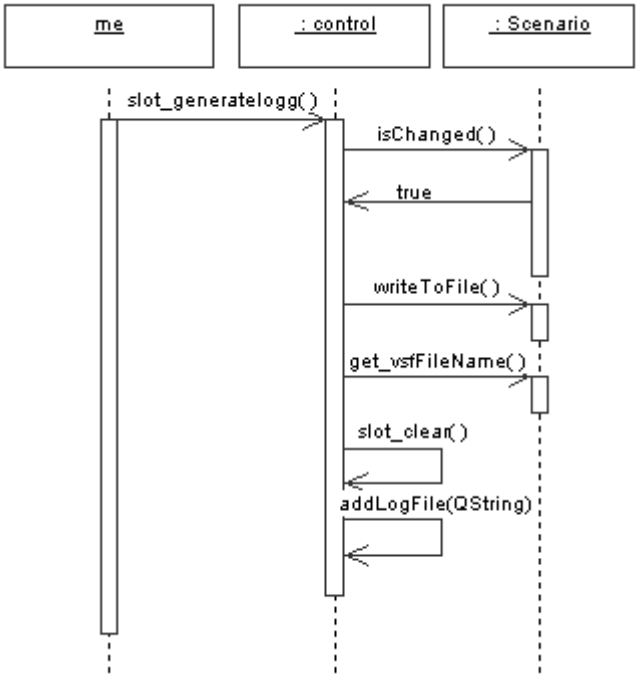
C.2 Save Scenario



C.3 Generate Log (no changes in dat files)



C.4 Generate Log (saving changes in dat files)



D Testresultat gentemot kravspecifikationen

Denna bilaga visar resultatet av de tester som gjorts på applikationen VisualSim 1.2 för att kontrollera att programmet uppfyller kraven i kravspecifikationen. Nedanstående punkter är därför relaterade till punkterna i kravspecifikationen. De punkter som av kravspecifikationen betecknats som "ej tillämpligt" har av uppenbara skäl ej heller testats. Testerna genomfördes när applikationen kändes stabil. Vi demonstrationskörde programmet för handledaren på SBD som kontrollerade att de krav som finns i kravspecifikationen uppfylldes.

D.1 Symbolbeskrivning

✓ kravet uppfyllt

✗ kravet ej uppfyllt

D.2 Testresultat

3.2 Kapabilitetskrav för VisualSim 1.2

3.2.1 Data

Krav:

a) Ett nytt filformat, vsf (VisualSim Scenario File), skall skapas för att hålla reda på vilken datfil (rotfilen) och loggfil som används vid ett scenario. Filen skall även innehålla kommentarer om scenariot. ✓

b) Efter att information matats in i dialogrutan som beskrivs i 3.2.2.2, skall informationen sparas i datfiler för att köras i DuellSim. ✓

c) Alla filer i ett visst scenario skall genom sitt filnamn visa vilket scenario de tillhör, t ex myScenario.dat, myScenario_aqu.dat, myScenario.dsl o s v. ✓

d) En importerad datfil, se 3.2.2.2 krav f, skall kopieras och få ett namn passande det scenario det importeras till. Ändringar som görs ✓

påverkar inte originalfilen. *Exempel*: *Om filen myScenario_aqu.dat importeras till scenariot yourScenario, skall en ny fil yourScenario_aqu.dat skapas som är en exakt kopia av myScenario_aqu.dat.*

e) Vid import av en datfil skall även datfiler länkade från denna fil importeras och döpas om enligt punkt d. ✓

f) Rotfilen skall ej gå att importera. Istället skall menyalternativet ”spara som” finnas för scenarion, vilket ger samma resultat. ✓

g) Loggfiler skall genereras med hjälp av den information som finns i scenariots nyskapade datfiler. Genereringen skall ske med DuellSim. ✓

h) En fil innehållande sökvägen till Duellsim skall läsas in vid uppstart. Vid ändring av sökväg skall denna fil uppdateras. ✓

Bör:

a) En loggfil som genererats ska automatiskt laddas in. ✓

b) Vid import av en datfil skall endast rätt sorts datfiler visas i dialogrutan. *Exempel: Man vill importera en AQU datfil. Endast de datfiler som innehåller data för AQU’n kommer att visas.* ✓

3.2.2 Användargränssnitt

3.2.2.1 Meny

Krav:

a) I Arkivmenyn skall val läggas till för att ✓

- skapa scenario
- öppna scenario
- spara scenario
- spara scenario som
- redigera scenario
- stänga scenario
- generera loggfil
- sätta sökväg till DuellSim

Bör:

a) Valen ovan skall även finnas som snabbvalsknappar under menyn. ✗

Anmärkning:

a) Detta krav realiserades inte p g a problem att få ordning på koden i det arbete som vi vidareutvecklat.

3.2.2.2 Dialogruta för inmatning av indata för scenario

Krav:

a) En dialogruta med flikar skall användas för att enkelt kunna hoppa mellan inmatning av de olika aktörernas data. Flikarna skall innehålla fält för inmatning av data för: ✓
   - data som sparas i rotfilen, flik General
   - spaningssensor, flik AQU
   - eldledningsenhet, flik FCU
   - stridsledning inklusive robotar, flik CCC
   - mål, flik Targets

b) All information om objekts egenskaper (såsom positioner, hastigheter), som idag finns lagrade i datfilerna (d v s indatan), skall kunna matas in manuellt av användaren genom ovan beskrivna dialogruta. ✓

c) För varje värde i dialogrutan skall vid skapandet av ett nytt scenario finnas ett defaultvärde hämtade från ”default”-datfiler. ✓

d) För vissa data skall användaren endast kunna mata in värden mellan en min och max gräns. ✓

e) Vid inmatning av ogiltigt värde skall programmet meddela detta, och datan ifråga skall sättas till defaultvärde. ✓

f) Vid respektive flik skall användaren kunna importera redan färdiga datfiler. ✓

3.2.2.2 Presentation av luftduell

Bör:

a) Alla objekt på ritytan i den nuvarande presentationen av luftduellen bör göras om till 3D. ✗

b) 3D-presentationen bör kunna roteras. ✗

c) 3D-presentationen bör vara zoombar. ✗

3.5 Krav på anpassning

Krav:

a) Programmet skall kunna köras på en PC med operativsystemet Windows NT. ✓

Bör:

a) Programmet bör kunna köras på en arbetsstation SUN Solaris. Ej testat

b) Programmet bör kunna köras på en PC med operativsystemet Linux. Ej testat

3.11 Samspel människa – maskin

Krav:

a) Användargränssnittet skall vara användarvänligt. ✓

Anmärkning:

a) Vår handledare på SBD ansåg att detta krav var uppfyllt eftersom mer arbete än nödvändigt lagts ner på utformning och användarhjälp.

3.13 Programmeringsspråk

Krav:

a) Uppbyggnaden av programmet skall vara objektorienterat och programmet skall implementeras i C++ med verktyget Qt 3.0. Koden kompileras med Microsoft Visual C++ 6 kompilatorn. ✓

b) Implementationen skall vara lättförståelig. ✓

c) Koden till programmet skall vara väl kommenterad. ✓

E Black Box test

Testfallen i denna bilaga har definierats utifrån olika scenarion och användarfall som kan tänkas uppstå under exekvering av VisualSim. Vi har dock inte tagit upp alla tänkbara användarfall, utan endast ett par basfall som kan tänkas ställa till problem för systemet och som vi tyckt varit viktiga att testa.

E.1 Cancel-knapparnas funktionalitet

E.1.1 Beskrivning

Om en ändring görs i en dialogruta och användaren klickar på Cancel så skall inga ändringar sparas, samt att programmet inte skall anse att ändringar gjorts. Dialogrutorna finns i flera nivåer, t ex när man klickar på "edit target" får man upp ytterligare en dialogruta, och oavsett på vilken nivå ändringar har gjorts så skall cancelfunktionen fungera, och alla värden skall sättas tillbaka till vad de var innan ändringarna gjordes.

E.1.2 Tillvägagångssätt

Testningen görs genom att ändra alla möjliga inmatningsfält, ett i taget, och därefter klicka på Cancel-knappen. Vid de dialogrutor som ligger på en högre nivå testas dels att klicka på deras Cancel-knapp, dels att klicka på deras Ok-knapp och sedan på "förälderns" Cancel-knapp.

E.1.3 Förväntat resultat

Efter att man klickat på Cancel förväntas alla inmatningsfält få tillbaka de värden som de hade innan ändringar gjordes.

E.1.4 Faktiskt resultat

Det faktiska resultatet motsvarade det förväntade resultatet.

E.2 Gränsvärden i inmatningsfält

E.2.1 Beskrivning

För att ett scenario skall återspegla verkligheten så mycket som möjligt, och för att simuleringen inte skall få fel indata, så måste inmatningsfälten ha gränsvärden.

E.2.2 Tillvägagångssätt

För att testa gränsvärden så testar vi så kallade "kritiska värden". Med detta menas att om ett värde har en gräns vid t ex 100, så testas värdena 99, 100 och 101. Om resultaten för dessa kritiska värden är detsamma som de förväntade resultaten, så kan man anta att gränsvärdeskontrollerna även fungerar för övriga värden.

E.2.3 Förväntat resultat

Om ett felaktigt värde matas in så skrivs ett felmeddelande ut och det tidigare värdet sätts tillbaka i inmatningsfältet.

E.2.4 Faktiskt resultat

Det faktiska resultatet motsvarade det förväntade resultatet.

E.3 Generera en loggfil

E.3.1 Beskrivning

När användaren har skapat ett scenario, så kan han välja att generera en loggfil för att låta VisualSim spela upp scenariot. Innan man genererar en loggfil måste scenariot vara sparat. Man kan dock inte generera en loggfil från defaultscenariot utan man måste i så fall spara om scenariot med ett annat namn. För att generera en loggfil krävs också att sökvägen till DuellSim är satt korrekt. Efter att en loggfil genererats skall den automatiskt läsas in i VisualSim, redo att spelas upp.

E.3.2 Tillvägagångssätt

I testet kommer vi att försöka generera en loggfil från defaultscenariot. Vi kommer också att se vad som händer om sökvägen till DuellSim inte är korrekt, samt kolla att en genererad loggfil läses in korrekt till VisualSim.

E.3.3 Förväntat resultat

Om en användare försöker att generera en loggfil från defaultscenariot så kommer ett felmeddelande att skrivas ut och användaren tillfrågas om han vill spara scenariot med ett annat namn. Om sökvägen till DuellSim inte är korrekt så skrivs ett felmeddelande ut med ett förslag till användaren att kolla sökvägen. Efter att en loggfil genererats kommer spellistan att tömmas och loggfilen läggas in först i listan redo att spela upp scenariot.

E.3.4 Faktiskt resultat

Det faktiska resultatet motsvarade det förväntade resultatet.

E.4 Spara scenario

E.4.1 Beskrivning

Ett scenario kan tänkas sparas av olika anledningar. Som nämnts i E.3.1 så måste ett scenario sparas innan en loggfil kan genereras. En annan anledning kan helt enkel vara att en användare vill spara ett scenario bara för att kunna jobba vidare med det vid ett senare tillfälle. Filändelsen för scenarion är vsf (VisualSim Scenario File). När scenariot sparas skall datfiler skapas med namn som förknippar filerna med rätt scenario, t ex myScenario.dat, myScenario_aqu.dat o s v.

E.4.2 Tillvägagångssätt

Vi kommer att försöka spara ett scenario med samma namn som defaultscenariot, d v s "default.vsf". I testet ingår också att försöka spara ett scenario med ett redan befintligt scenariofilnamn. Vi kommer även att bekräfta att korrekta datfiler har skapats. Detta görs dels genom att titta på filerna i en texteditor, och dels genom att stänga ett sparat scenario och öppna det igen. På så sätt ser man om värdena är desamma som innan man stängde scenariot.

E.4.3 Förväntat resultat

Om användaren försöker skriva över defaultscenariofilen, d v s spara filen som "default.vsf", så kommer ett felmeddelande att skrivas ut och användaren får möjlighet att ändra filnamn. Om filnamnet som användaren väljer att spara sitt scenario som redan finns, kommer detta att meddelas, användaren blir tillfrågad om den gamla filen skall skrivas över. VisualSim förväntas också skapa datfiler med namn som förknippas med scenariot.

E.4.4 Faktiskt resultat

Det faktiska resultatet motsvarade det förväntade resultatet.

E.5 Importera datfil och spara scenario

E.5.1 Beskrivning

Om användaren importerar en datfil till sitt scenario och sedan sparar scenariot så skall den importerade filen först kopieras, kopian skall sedan sparas med ett namn som kopplar filen till scenariot. Exempel: A_aqu.dat importeras till scenario B. När B sedan sparas kopieras A_aqu.dat och döps om till B_aqu.dat. De datfiler som går att importera är aqu- , ccc-, fcu-, robot-, och targetdatfiler.

E.5.2 Tillvägagångssätt

Vi kommer att importera datfiler vid alla de ställen i VisualSim där det är möjligt. Sedan sparas scenariot dit vi importerat datfilerna. Vi kommer sedan att först ta reda på om en fil har skapats för varje importerad fil, med korrekt filnamn. Sedan kommer vi att jämföra den kopierade filen med originalet för att säkerställa att det är en exakt kopia av den importerade filen. Jämförelserna görs i en vanlig texteditor.

E.5.3 Förväntat resultat

När användaren sparar ett scenario med importerade datfiler, så kommer exakta kopior att finnas för varje importerad fil. Dessa kopior kommer att ha ett namn som tydligt förknippas med det scenario de tillhör. Vid en jämförelse mellan kopian och originalet så skall innehållet filerna vara identiska.

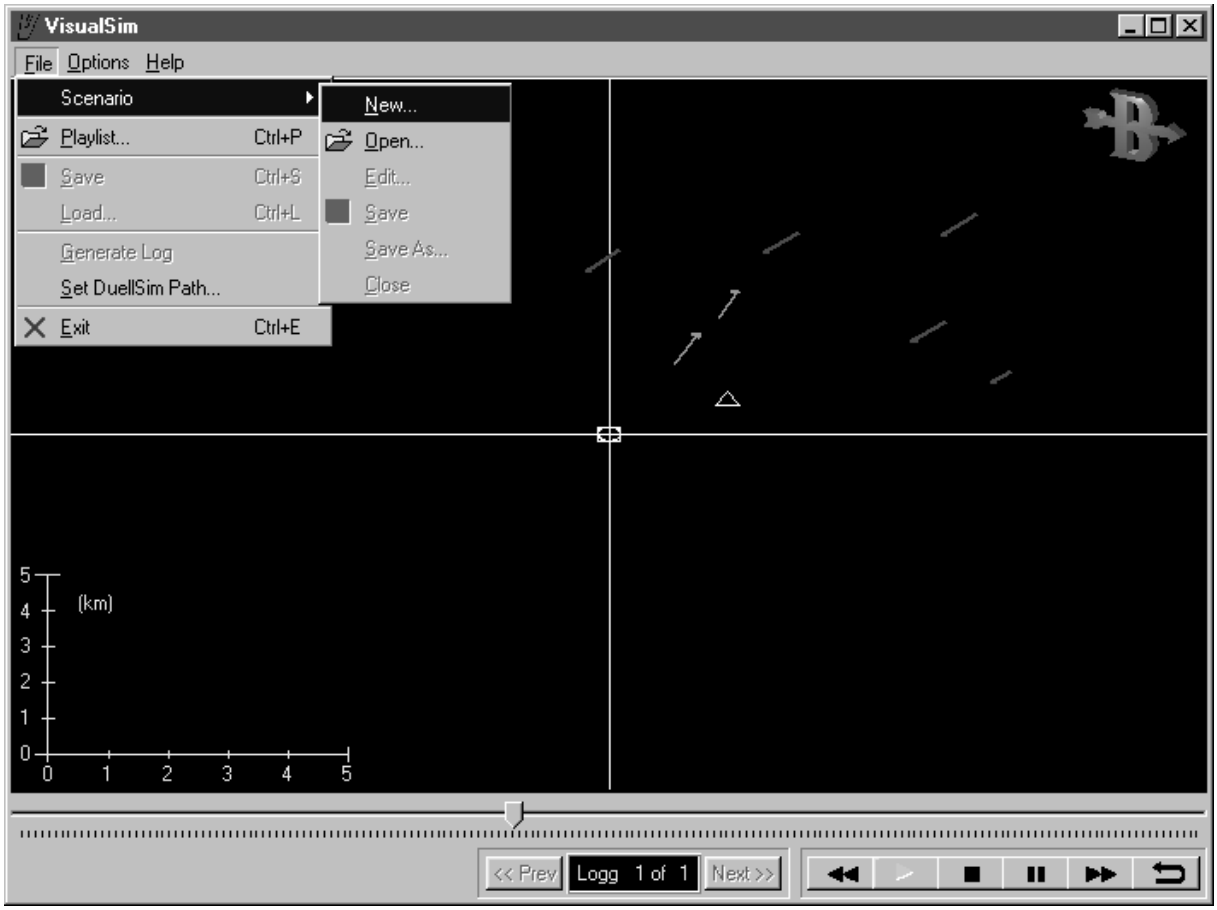
E.5.4 Faktiskt resultat

Det faktiska resultatet motsvarade det förväntade resultatet.

F Skärmdumpar

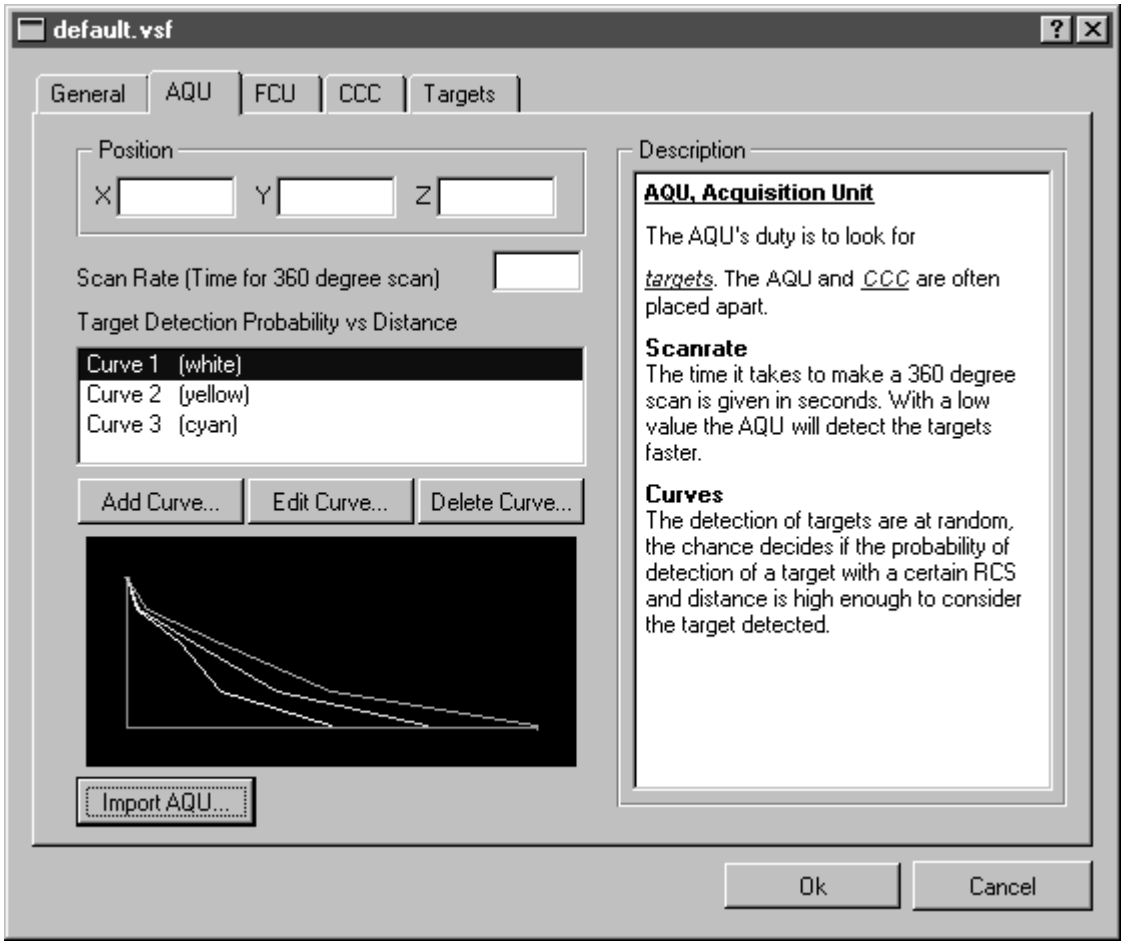
Observera att alla värden i dialogrutorna har raderats p g a sekretesskäl.

F.1 VisualSim 1.2



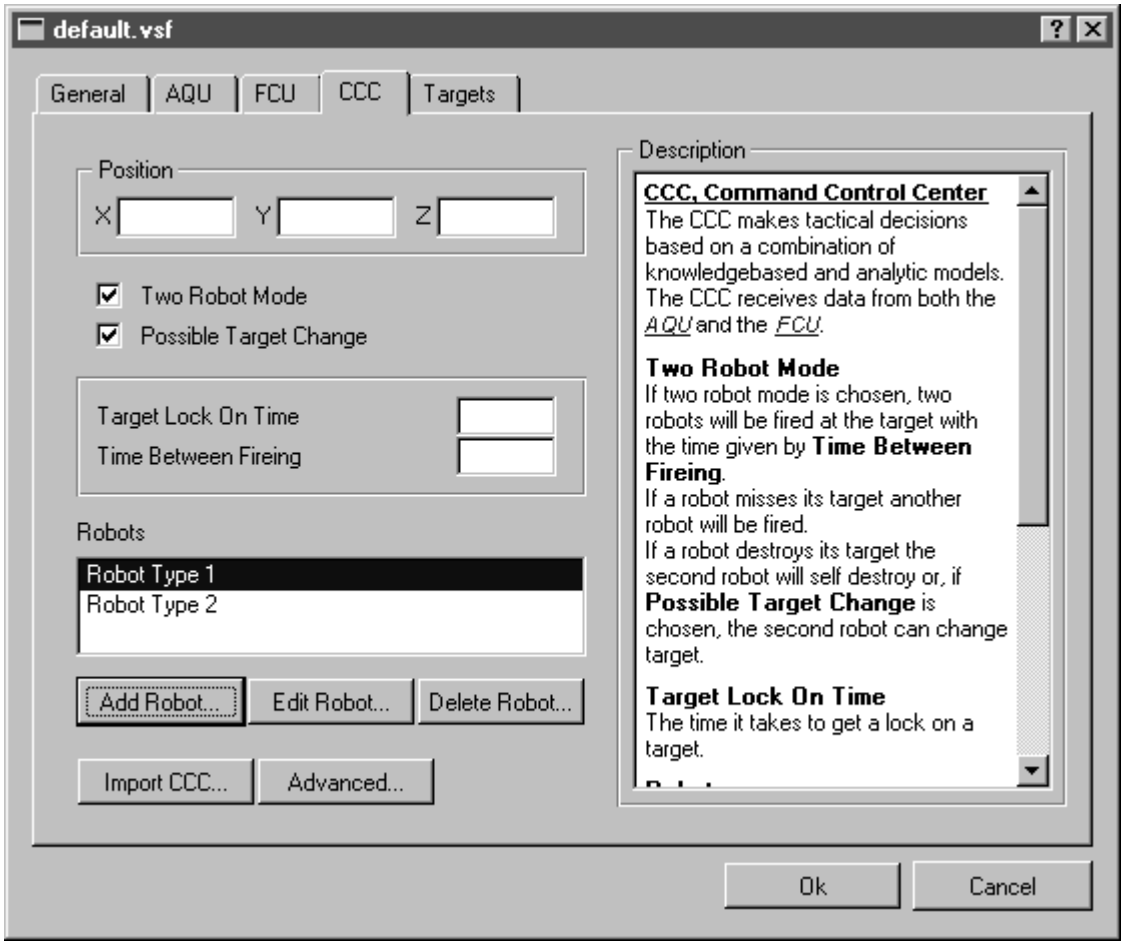
Bilden visar hur VisualSim ser ut efter vidareutvecklingen. Det som lagts till jämfört med figur 2.4 är nya menyval och 3D loggan.

F.2 Fliken AQU



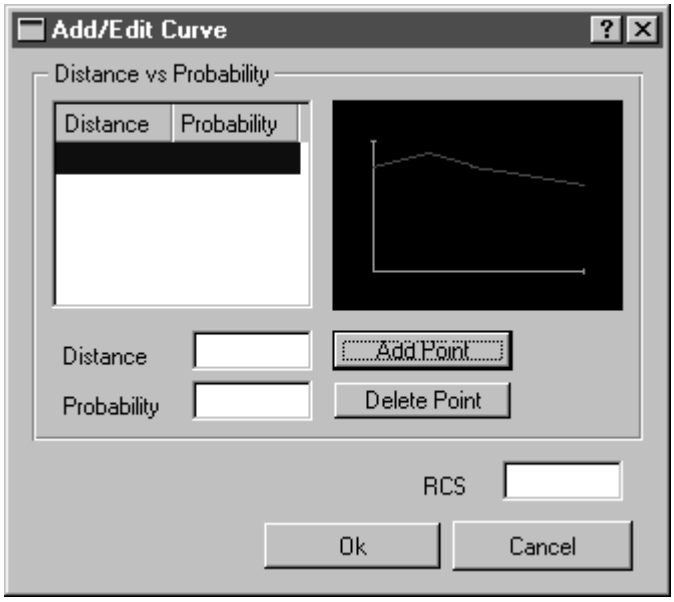
Bilden visar fliken AQU där användaren matar in data gällande just spaningssensorn. Till höger i dialogrutan finns en liten användarhjälp i html-format. Även kurvorna som visas är för att hjälpa användaren att tolka de värden som matats in.

F.3 Fliken CCC



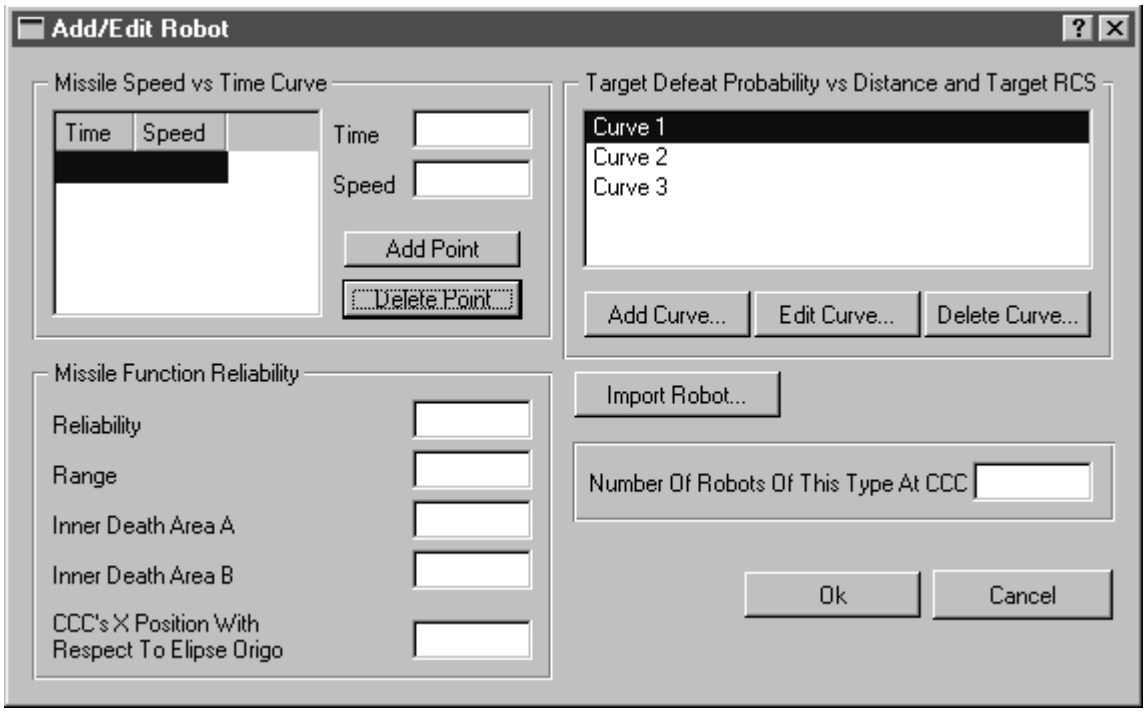
Bilden visar utseendet för fliken CCC där data för stridsledningen matas in. Även data för robotar styrs från denna flik. Alla flikar har ett liknande utseende, med inmatningsfält och en html-hjälp till höger. Detta är för att användaren skall känna igen sig.

F.4 Dialogrutan Add/Edit Curve



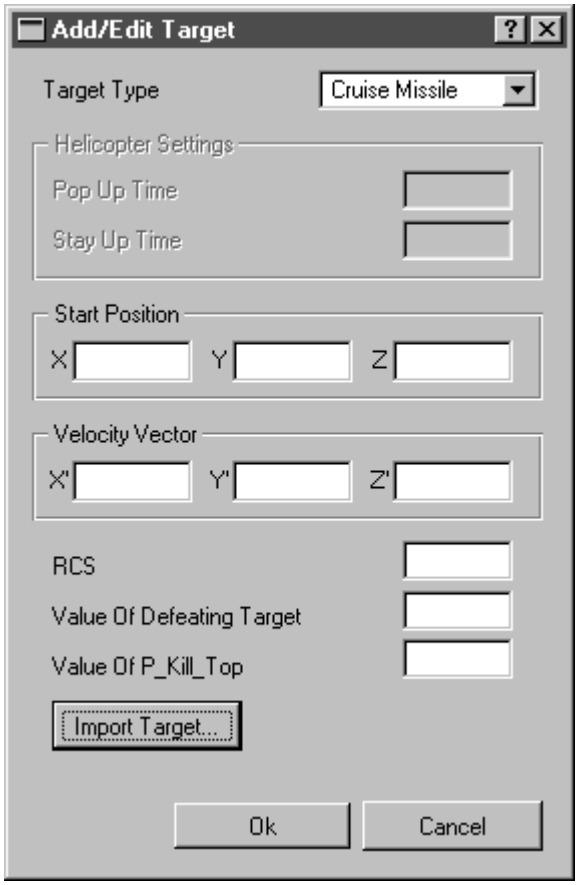
Bilden visar dialogrutan för inmatning av punkter i en kurva. Denna kurva visas även grafiskt. Det är bl a sådana kurvor som syns på fliken AQU som visas i F.2 som redigeras.

F.5 Dialogrutan Add/Edit Robot



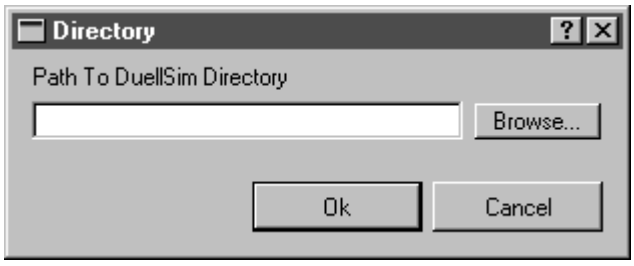
Bilden visar dialogrutan för inmatning av värden som styr robotarna.

F.6 Dialogrutan Add/Edit Target



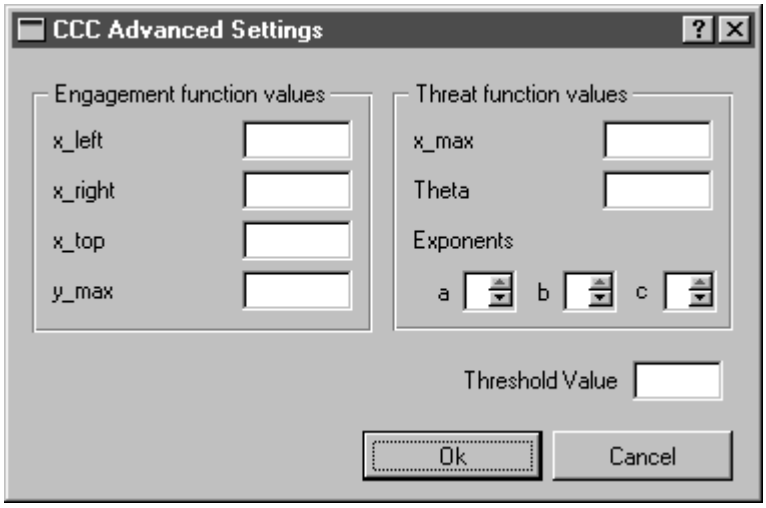
Bilden visar dialogrutan som används vid redigering av data för mål.

F.7 Dialogrutan Directory



Bilden visar dialogrutan som används för att sätta sökvägen till DuellSim.

F.8 Dialogrutan Ccc Advanced Settings



Bilden visar dialogrutan som används för avancerade inställningar för stridsledningen.