FOI-R--0906--SE
Juni 2003
ISSN 1650-1942
Metodrapport

Adolf Rosander

Statusberäknande modul mellan helikopter och hotsystem i en simuleringsprogramvara

Ledningssystem
581 11 Linköping

TOTALFÖRSVARETS FORSKNINGSINSTITUT

Ledningssystem
Box 1165
581 11 Linköping

FOI-R--0906--SE

Juni 2003

ISSN 1650-1942

Metodrapport

Adolf Rosander

Statusberäknande modul mellan helikopter och hotsystem i en simuleringsprogramvara

Utgivare	**Rapportnummer, ISRN**	**Klassificering**
Totalförsvarets Forskningsinstitut - FOI	FOI-R--0906--SE	Metodrapport
Ledningssystem	**Forskningsområde**	
Box 1165	6. Telekrig	
581 11 Linköping	**Månad, år**	**Projektnummer**
	Juni 2003	E7015
	Verksamhetsgren	
	5. Uppdragsfinansierad verksamhet	
	Delområde	
	61 Telekrigföring med EM-vapen och skydd	

Författare/redaktör	**Projektledare**
Adolf Rosander	**Godkänd av**
	Uppdragsgivare/kundbeteckning
	Tekniskt och/eller vetenskapligt ansvarig

Rapportens titel

Statusberäknande modul mellan helikopter och hotsystem i en simuleringsprogramvara

Sammanfattning (högst 200 ord)

Institutionen för Telekrigvärdering vid Totalförsvarets forskningsinstitut, FOI i Linköping, fick 1999 i uppdrag att göra en studie om VMS, Varnar och Motverkan System för helikopter. Studien skulle bland annat resultera i en simuleringsprogramvara ”Terrängmodell Upptäckt”. Den här modellen skulle användas för att simulera en helikopters risk för upptäckt och bekämpning längs en flygbana i en 3D-terräng.

Den här rapporten presenterar arbetet med att designa och implementera den del av ”Terrängmodell Upptäckt”, som beräknar och presenterar vilken status helikoptern har i förhållande till specificerade hotsystem, när helikoptern flyger längs en bana i 3D-terrängen.

Rapporten börjar med en genomgång av vilka krav som ställts, följt av en analys av vilka komponenter som behövs samt hur gränssnittet ska utformas. Sedan följer beskrivning av implementeringen och det uppnådda resultatet.

Simuleringsprogramvaran utgörs av ett C++ program kopplat mot 3D-motorn Vega. Beräkningar för att bestämma status för helikoptern utgår från siktlinjesberäkningar och systemtider som fås från simuleringsprogramvaran. Vilken status helikoptern har i varje punkt presenteras av ett färgat spår efter helikoptern i 3D-visualiseringen.

Resultatet av examensarbetet är en implementering av statusberäkningen samt presentationen av vilken status helikoptern har i simuleringsprogramvaran ”Terrängmodell Upptäckt”. Exempel på en körning av simuleringsmodellen finns i Appendix A.

Nyckelord

VMS, beräkningsmodul, helikopter, helikopterstatus, Vega, Telekrigsimulering

Övriga bibliografiska uppgifter	**Språk** Svenska
ISSN 1650-1942	**Antal sidor:** 43 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Issuing organization
FOI – Swedish Defence Research Agency
Command and Control Systems
P.O. Box 1165
SE-581 11 Linköping

Report number, ISRN
FOI-R--0906--SE

Report type
Methodology report

Programme Areas
6. Electronic Warfare

Month year
June 2003

Project no.
E7015

General Research Areas
5. Commissioned Research

Subcategories
61 Electronic Warfare including Electromagnetic Weapons and Protection

Author/s (editor/s)
Adolf Rosander

Project manager

Approved by

Sponsoring agency

Scientifically and technically responsible

Report title (In translation)
State calculating module between helicopter and threatsystem in a simulation software

Abstract (not more than 200 words)
The department of Electronic Warfare Assessments at the Swedish Defence Research Agency (FOI) in Linköping, 1999 got the commission to make a study about "VMS", Warning and Countermeasure System for helicopter. The study would among other things result in simulation software "Terrängmodell Upptäckt". This simulation software would be used to run simulations where the helicopters risk for detection and being shoot down is studied.

This thesis presents the work with the design and the implementation of that part of the "Terrängmodell Upptäckt" which calculates and presents what state the helicopter has in relation to specified threat systems during the helicopters flight through the 3D-terrain.

The simulation software consists of a C++ program connected to Vega. Vega is a COTS product containing a 3D-engine and a tool for scene generating. Calculations to decide the state of the helicopter are made from line-of-sight calculations and system times. What state the helicopter has in each coordinate is presented by a colored trail behind the helicopter in the 3D-visualization.

The result of the thesis is the implementation of the state calculating part and the presentation of what state the helicopter has. Example of simulation runs can be found in appendix A.

Keywords
VMS, helicopter, helicopterstate, Vega, Electronic Warfare Assessments

Further bibliographic information

Language Swedish

ISSN 1650-1942

Pages 43 p.

Price acc. to pricelist

Abstract

The department of Electronic Warfare Assessments at the Swedish Defence Research Agency (FOI) in Linköping, 1999 got the commission to make a study about "VMS", Warning and Countermeasure System for helicopter. The study would among other things result in a simulation software "Terrängmodell Upptäckt". This simulation software would be used to run simulations where the helicopters risk for detection and being shoot down is studied.

This thesis presents the work with the design and the implementation of that part of the "Terrängmodell Upptäckt" which calculates and presents what state the helicopter has in relation to specified threatsystems during the helicopters flight through the 3D-terrain.

The simulation software consists of a C++ program connected to Vega. Vega is a COTS product containing a 3D-engine and a tool for scene generating. Calculations to decide the state of the helicopter are made from line-of-sight calculations and system times. What state the helicopter has in each coordinate is presented by a colored trail behind the helicopter in the 3D-visualization.

The result of the thesis is the implementation of the state calculating part and the presentation of what state the helicopter has. Example of simulation runs can be found in appendix A.

Sammanfattning

Totalförsvarets forskningsinstitut, FOI i Linköping, fick 1999 i uppdrag att göra en studie om VMS, Varnar och Motverkan System för helikopter. Institutionen för Telekrigvärdering tog sig an uppgiften. Studien skulle bland annat resultera i en simuleringsprogramvara ”Terrängmodell Upptäckt”. Den här modellen skulle användas för att simulera en helikopters risk för upptäckt och bekämpning längs en flygbana i en 3D-terräng.

Den här rapporten presenterar arbetet med att designa och implementera den del av ”Terrängmodell Upptäckt”, som beräknar och presenterar vilken status helikoptern har i förhållande till specificerade hotsystem, när helikoptern flyger längs en bana i 3D-terrängen.

Rapporten börjar med en genomgång av vilka krav som ställts, följt av en analys av vilka komponenter som behövs samt hur gränssnittet ska utformas. Sedan följer beskrivning av implementeringen och det uppnådda resultatet.

Simuleringsprogramvaran utgörs av ett C++ program kopplat mot Vega. Vega är en kommersiell produkt som består av en 3D-motor och ett scengenereringsverktyg med tillhörande API. Beräkningar för att bestämma status för helikoptern utgår från siktlinjesberäkningar och systemtider som fås från simuleringsprogramvaran. Vilken status helikoptern har i varje punkt presenteras av ett färgat spår efter helikoptern i 3D-visualiseringen.

Resultatet av examensarbetet är en implementering av statusberäkningen samt presentationen av vilken status helikoptern har i simuleringsprogramvaran ”Terrängmodell Upptäckt”. Exempel på hur en körning av simuleringsmodellen resulterar i olika status och hur dessa presenteras finns i appendix A.

Förord

Jag vill tacka min handledare Lars Tydén samt alla andra från FOI inblandade som gjort mitt arbete intressant, givande och trevligt. Jag vill även tacka min examinator Patrick Doherty.

Innehållsförteckning

Figur och tabellförteckning

1 Inledning

Den här rapporten har skrivits inom ramen för ett 10-poängs examensarbete. Uppdragsgivare för examensarbetet är FOI.

1.1 Notationer och begrepp

VMS = Varnar och motverkansystem

I texten är källor refererade enligt Oxford Reference System. Hakparanteser används ([1] första källan) för att markera en källa, på samma sätt finns respektive källa representerad i referenskapitlet.

Ordförklaringar återfinns i appendix C.

1.2 Bakgrund och problemformulering

Med anledning av svenska försvarets framtida inköp av VMS till helikopter fanns det intresse hos Försvarsmakten att bygga upp kunskap och kompetens inom området. Studien ”VMS för helikopter” har genomförts på uppdrag av HKV KRI Plan (Högkvarteret Krigsorganisationledning Planering). Studiegruppen har i huvudsak bestått av medlemmar från Helikopterflottiljen samt forskare från FOI. Framtagandet av två simuleringsmodeller ”Terrängmodell Upptäckt” och ”Fackelfällarmodellen” var en mycket väsentlig del av studieförsöket. Formulerade krav på modellerna redovisades i delrapport för 2001 [2]. Arbetet utgör en viktig delkomponent i anskaffningen och vidareutvecklingen av helikoptersystem.

Den ena av simuleringsmodellerna, ”Terrängmodell Upptäckt”, konstruerades för att i första hand utnyttjas för studier av risken för upptäckt och bekämpning i valt terrängavsnitt. I simuleringsmodellen finns 3D-modeller för terräng, hotsystem och helikopter. I 3D-världen finns specificerade hotsystem utplacerade. De hotsystem som specificeras är spaningsradar, automatkanon, radarrobot, pansarvagnsrobot, IR-robot av bildalstrande typ samt IR-robot av typ retikel.

Utifrån generiska prestanda för hotsystemens sensorer används tidsuppmätningar (från simuleringsprogramvaran) för att kalkylera status för helikoptern. Detta genomförs genom att en helikopter åker längs en bana i 3D terrängen och avdömningen för upptäcktsrisk och bekämpningsrisk sker dynamiskt allt eftersom helikopterns position förändras längs helikopterbanan.

Examensarbetet består i att konstruera den avdömningsmodul i demonstrationsprogrammet ”Terrängmodell Upptäckt” som med hjälp av 3D-beräkningar, systemparametrar och systemtider kontinuerligt gör statusavdömningen samt presenterar resultatet.

1.3 Syfte

Syftet med examensarbetet är:

- Analysera vad som krävs för att lösa den givna uppgiften
- Designa beräkningsmodulen
- Konstruera ett intuitivt grafiskt gränssnitt till beräkningsmodulen

- ? Implementering av beräkningsmodulen och det grafiska gränssnittet
- ? Skriva en rapport som beskriver förfarandet

1.4 Omfattning

Uppgiften består i att designa och implementera modellen för vilken status helikoptern har i förhållande till spanings- och vapensystemen samt att presentera detta som ett färgkodat spår bakom helikoptern i 3D visualiseringen. Färger skall vara valbara via ett grafiskt gränssnitt.

1.5 Avgränsningar

Utplacering av vapen och spaningssystem ingår inte i uppgiften. Likaså ingår ej konstruerande av flygbanor.

1.6 Metod

Programmet för att demonstrera ”Terrängmodell upptäckt” modellen programmeras i Visual C++ [3] som använder sig av Vega [4] för att visualisera och organisera 3D-modellerna. Vega är en COTS produkt som består av 3D-motor och scengenereringsverktyg med ett API som har god dokumentation. Koordinater för helikopter och andra dynamiska objekt erhålls av Vega. Istället för att utnyttja Vegas inbyggda funktioner för att få fram vektorer mellan olika koordinater i 3D-världen utförs sådana geometriska beräkningar i C++ för att uppnå större kontroll. Efter detta används Vega (via API anrop) för att hitta kollisionskoordinater mellan vektorer och volymer (siktlinjesberäkningar). Som designverktyg har Rational Rose [5] använts.

Valet av att designa klasserna med UML [6] i Rational Rose grundar sig på att UML är en vedertagen standard samt att Rational Rose har goda funktioner för att överföra design till klasser i Visual C++. Rational Rose stödjer även ”Reverse engineering” som utifrån kod genererar klassdiagram i UML.

Det designmässiga förfarandet kommer i stor utsträckning att influeras av egen programmeringsstil, vana och kunskap. Den här uppgiften är relativt unik och färdiga lösningsförslag kommer därför att bli svåra att finna. Objektorienterad design kommer att eftersträvas. Eftersom objektorienterad design tillåter skilda lösningar kommer viss del av designen att diskuteras fram med handledare. Dock kommer de viktigaste besluten i designförfarandet att diskuteras i denna rapport. Återanvändande av kod kommer att ske i möjligaste mån.

Några små jämförande studier kommer att utföras mellan de olika alternativ som finns för GUI-delen av projektet.

Följande punkter skall fördelas på de 10 veckor som detta examensarbete är beräknat att behöva. Tidsåtgången för varje punkt uppskattas ej utan fördelningen mellan punkterna justeras dynamiskt under arbetets gång.

1. Bygga upp förståelse för uppgiften och programmiljön
2. Analys och design
3. Implementering av logikdel

4. Implementering av GUI
5. Testning
6. Rapportskrivning

1.7 Källor

Eftersom det här examensarbetet är av praktisk art kommer stor del av litteraturhänvisningarna att beröra programmering/implementering. Många sådana källor fås från webben för att få tillgång till den senaste informationen.

1.8 Struktur på rapporten

Strukturen på rapporten har konstruerats utifrån en eftersträvan att uppnå en metodisk genomgång av arbetet med examensarbetet.

1.8.1 Kravspecifikationskapitel

Rapporten börjar med ett kapitel som berör utvalda delar av den kravspecifikation som lagt grunden för uppgiften. I kapitlet redogörs för mål, krav och premisser för uppgiften.

1.8.2 Analyskapitel

I analyskapitlet ställs inledningsvis frågor som bör besvaras av analyskapitlet. Därefter följer analys av logikprogrammeringen och analys av GUI. Här är vissa mindre jämförande studier gjorda för att komma fram till vilka lösningar som skall användas. Analyskapitlet avslutas med att de inledande frågorna besvaras.

1.8.3 Designkapitel

I designkapitlet jämförs två olika designförslag av logikdelen.

1.8.4 Implementeringskapitel

Beskrivning av den praktiska implementering av logikdel och GUI-del.

1.8.5 Resultatkapitel

Resultatkapitlet börjar med en beskrivning av hur programvaran fungerar. Resultatkapitlet avslutas med att återknyta till vilka krav och målsättningar från kravspecifikationen som uppnåtts.

1.8.6 Diskussionskapitel

I diskussionskapitlet diskuteras hur modulen slutligen kom att användas samt vad som kunde gjorts annorlunda/bättre. Här finns också delar som berör möjliga förbättringar och ytterligare framtida användning.

1.8.7 Appendix

Appendix A: Körningsexempel av modulen
Appendix B: Körning av scenarion i den applikationen där modulen slutligen kom att användas.
Appendix C: Här finns några utvalda förklaringar till ord och begrepp.

2 Kravspecifikation

I detta kapitel berörs utvalda delar av den kravspecifikation som lagt grunden för uppgiften. De delar som utelämnas har inte varit relevanta för förståelsen för uppgiften och dess lösning vilka beskrivs i den här rapporten.

2.1 Mål

Designa och implementera modellen för vilken status helikoptern har i förhållande till spanings- och vapensystemen samt att presentera detta som ett färgkodat spår bakom helikoptern i 3D-visualiseringen (se figur 2.1). Färger för de status som finns skall kunna väljas i ett grafiskt gränssnitt.

Figur 2.1 Helikopterns status presenteras i form av ett spår efter helikoptern

2.2 Krav

Färger skall vara valbara. Exempel på färger och status finns i tabell 2.1.

Tabell 2.1 Exempel på färgkodning av spåret

Status	Färgkod
Inget hot	Grön
Upptäckt av spaningsradar	Orange
IR-robot avfyrning möjlig	Ljusblå
Automatkanon avfyrning möjlig	Blå
IR-robot bekämpning möjlig	Rosa
Automatkanon bekämpning möjlig	Röd
...	...
Upptäckt (mer än ett system)	Gul
Avfyring möjlig (mer än ett system)	Violett
Bekämpning möjlig (mer än ett system)	Vit

Motmedel skall kunna användas i form av facklor och har betydelse för IR-robot. Spaningssystemet består i en spaningsradarstation som kan stå på valfri plats i 3D-terrängen. Radarstationen avgör om målet går att detektera i den position där målet befinner sig.

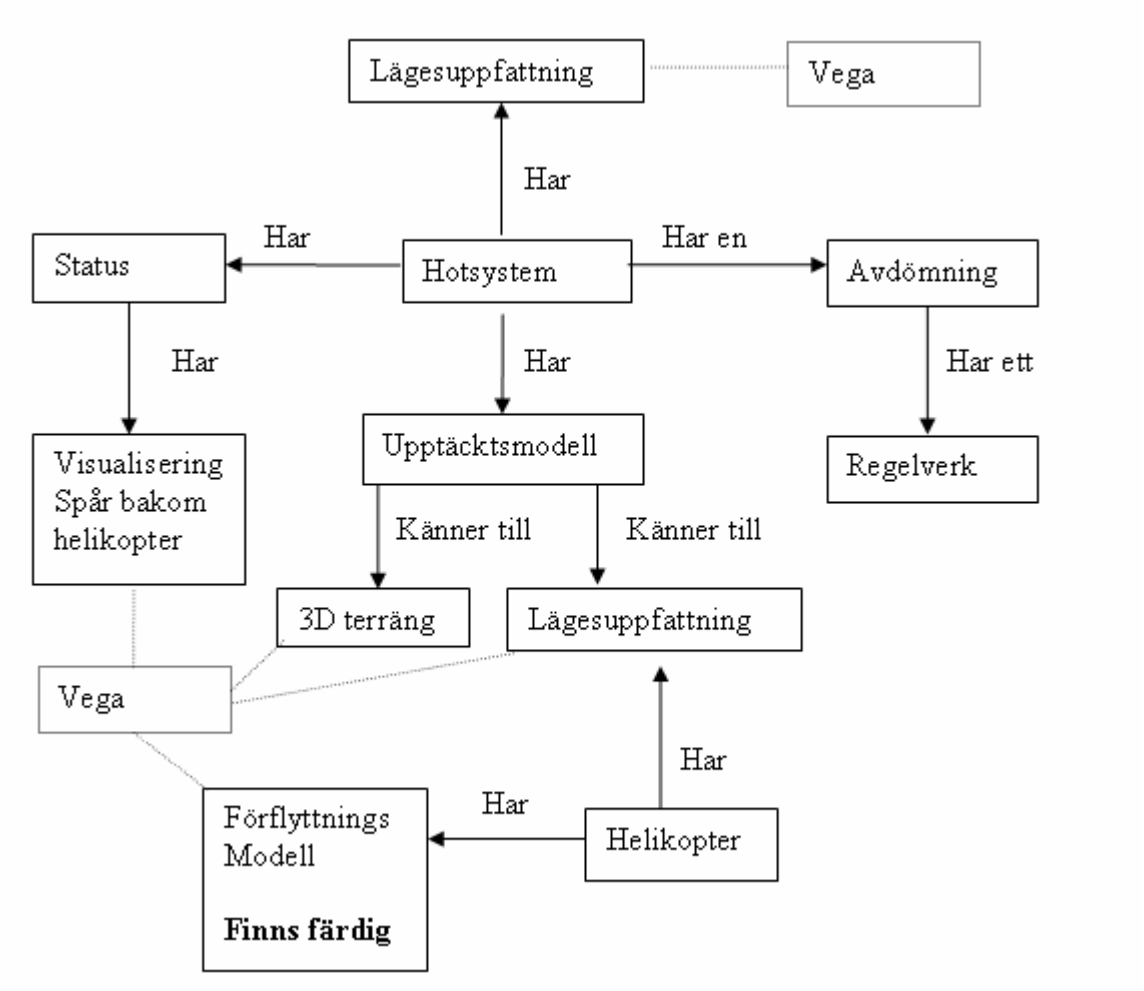
Följande vapensystem skall kunna avfyras från valfri plats i modellen:

- IR-robot = retikelmålsökare respektive bildalstrande robot med målsökare som använder infraröd strålning
- Radarrobot = robot med en radarmålsökare
- Automatkanon = här simuleras möjlighet till bekämpning efter invisning av radar och/eller sikte och med avseende på systemtider
- Pansarvagnsrobot = Kommandostyrd med tråd eller laserledstråle. Simuleras med avseende på systemtider

Avdömningen av helikopterns status utgår från systemtider som uppmäts i simuleringsprogramvaran samt siktlinjesberäkningar i 3D-landskapsmodellen. Allteftersom helikoptern förflyttar sig i terrängen sker avdömningen dynamiskt. Uppdateringen av status ska ske i realtid och därför finns även ett krav på att prestandan (uppdateringsfrekvensen) ej får understiga känslan av realtidssimulering.

2.3 Exempel på begreppsmodell

I figur 2.2 illustreras ett exempel på en begreppsmodell för det som uppgiften berör.



Figur 2.2 Översiktlig begreppsmodell

2.4 Premisser

Uppgiften löses i Visual C++ med hjälp av 3D-motorn Vega. Anledningen till att Vega skall användas är att det är en väldokumenterad motor med ett högnivå API, LynX. Har man tillgång till 3D-modeller för terräng och objekt kan man i LynX på några minuter konfigurera en 3D-värld med terräng, objekt med rörelsescheman, betraktningsvyer mm utan att skriva en enda rad programkod. En nackdel med Vega är att det på marknaden prestandamässigt sett finns snabbare 3D-motorer men dessa är ofta lågnivå3Dmotorer. Ytterligare en nackdel med Vega är den dyra licensen som måste betalas för varje dator som använder Vega. Här har FOI gjort en avvägning och kommit fram till att antalet datorer som kommer att använda den färdiga ”Terrängmodell Upptäckt” är så få att det inte är motiverat att lägga ut utvecklingspengar på en egen 3D-motor alternativt en billigare lågnivå 3D-motor. Anledningen till att Vega och LynX skall användas är alltså för att spara pengar på förkortat utvecklingsarbete.

En kortfattad beskrivning av hur helikopterns status ska beräknas för de olika hotsystemen.

Status: **Ej upptäckt**
Ingen av nedanstående status är uppfyllda.

Status: **Upptäckt av spaningsradar**
Fri sikt till helikoptern samt skall, sett från radarn, helikoptern inte ha terräng för nära i bakgrunden (specificerat antal meter) under en given tidsperiod.

Status: Helikopterbekämpning med IR-robot möjlig
Efter målangivelse från spaningssystem skall det vara fri sikt till helikoptern under en given tidsperiod. Om motmedel i form av facklor finns och används är helikoptern ej bekämpningsbar av IR-robot av typ retikel. Om bildalstrande målsökare används är helikoptern bekämpningsbar oavsett om helikoptern använder facklor eller ej.

Status: IR-robotbekämpning möjlig
Helikoptern är bestyckad med motmedel av typ facklor och IR-robot av typ retikel kan avfyras mot helikoptern. Det skall dessutom vara fri sikt till målet under beräknad transporttid för robot från möjlig avfyrning till kollision med helikopter.

Status: Helikopterbekämpning av radarrobot möjlig
Fri sikt till helikoptern samt ingen terräng ett antal meter mellan helikoptern och bakgrunden under given tid efter målangivelse från spaningssystem.

Status: Radarrobotbekämpning möjlig
Helikoptern är bestyckad med motmedel av typ remsor och hotsystem av typ radarrobot kan avfyras mot helikoptern. Det skall dessutom vara fri sikt till målet under beräknad transporttid för robot från möjlig avfyrning till kollision med helikopter.

Status: Automatkanonavfyrning möjlig
Fri sikt till helikoptern under given tid efter målangivelse från spaningssystem.

Status: Pansarvagnsrobotavfyrning möjlig
Fri sikt till helikoptern under given tid efter målangivelse från spaningssystem.

2.5 Nivåer
De olika nivåerna beskriver olika grad av implementering av uppgiften. Nivå A skall göras, nivå b, c, d implementeras i mån av tid.

A) Designa och implementera den givna uppgiften.
B) Utöka med en vinkelberoende räckvidd för hotsystemen där vinkeln är den vinkel som helikoptern har till hotsystemet. Utöka med att helikoptern förbrukar facklor när robotar avfyras mot den. Facklorna kan ta slut och därefter har helikoptern inga facklor.
C) Lägga till ett varnarsystem på helikoptern. Varnarsystemet varnar för inkommande robot efter en specificerad tidsrymd efter att en robot är avfyrad. Detta gäller under förutsättning att denna ligger inom systemets räckvidd och har fri sikt till roboten. Varnarsystemet kastar automatiskt ut facklor/remsor vid varning.
D) Göra en alternativ design som implementeras och därefter jämföra för- och nackdelar mellan de två olika designerna.

3 Analys

Det finns fördelar med att separera analys och implementering av logikdel och GUI till två skilda delar. Anledningen till detta är att de två delarna är fristående från varandra och på så sätt görs analys och implementering mer översiktlig.

3.1 Frågor som bör besvaras av analysen av uppgiften

Frågor som bör besvaras i analysen av uppgiften är följande:

1. Vad är huvuduppgiften?
2. Vilka klasser behövs i logikdelen?
3. Hur ska GUI utformas?
4. Vilka begränsningar finns och vilken prestanda kan förväntas krävas?
5. Vilka antaganden, uppdelningar och förenklingar kan/bör göras?

3.2 Förenklingar, uppdelningar och antaganden

Enligt kraven kan motmedel användas mot IR-robotar med retikel och ej bildalstrande IR-robotar. Detta betyder att hänsyn till typ erfordras vid behandlande av IR-robotar. För att kunna definiera hotsystemen mer generellt och därmed få ett mer enhetligt system separeras de två IR-robotarna till två separata hotsystem, retikel respektive bildalstrande. De två typerna kommer att behandlas som två helt separata typer av hotsystem. Alternativet är att införa en variabel som skiljer IR-robotarna åt men då riskeras att definieringen av ett hotsystem blir för komplex och svårbehandlad.

Efter en undersökning av Vega och dess möjligheter framkom att dess kollisionsdetektering av volymer är ofullständig. Detta riskerar att generera problem i implementeringen av siktlinjesberäkningar mot helikoptermodellen som helst bör representeras av en volym för att uppnå bästa realism. För att undvika eventuella problem utförs siktlinjesberäkningar mot en enda koordinat i helikoptermodellens centrum. När kollisionsdetektering av volymer färdigställs i Vega kommer uppdatering av modellen lätt kunna ske för att utnyttja denna. I stället för att förenkla förfarandet av siktlinjesberäkningar skulle en manuell implementering av volymkollisionsdetektering vara möjlig. En sådan implementering riskerar dock att bli beräkningskrävande.

3.3 Design av logikdelen

Programvaran som kontinuerligt utför beräkningen av status för helikoptern kommer att bestå av en cykel med exekverbar kod. Denna kodcykel som upprepar sig själv tills simuleringen är klar kallas huvudloopen. Efter varje varv i huvudloopen beräknas helikopterns status och uppdateringsfrekvensen ges alltså av hur lång tid det tar att exekvera ett varv i huvudloopen. Det är rimligt att anta att följande bör utföras i huvudloopen:

1. Uppdatera hotsystemen i förhållande till helikoptern.
2. Avdöma vilken status helikoptern får gentemot hotsystemen.
3. Sätta färgen på spåret utifrån statusen och rendera scenen
4. Uppdatera positioner, systemtider.

Intuitivt behövs följande klasser:
Hotsystem: Hanterar intern status, interna systemtider, position.
Helikopter: Hanterar intern status, position, antal facklor, remsor.
Avdömningsklass: Avdömer hotsystemens och helikopterns status.
Regelverk: Tillhandahåller de regler som avdömningsklassen behöver för att bestämma respektive status.
Vegaklass: Hanterar kopplingen mellan 3D-motorn och helikopter och hotsystem.

3.4 Analys av GUI delen

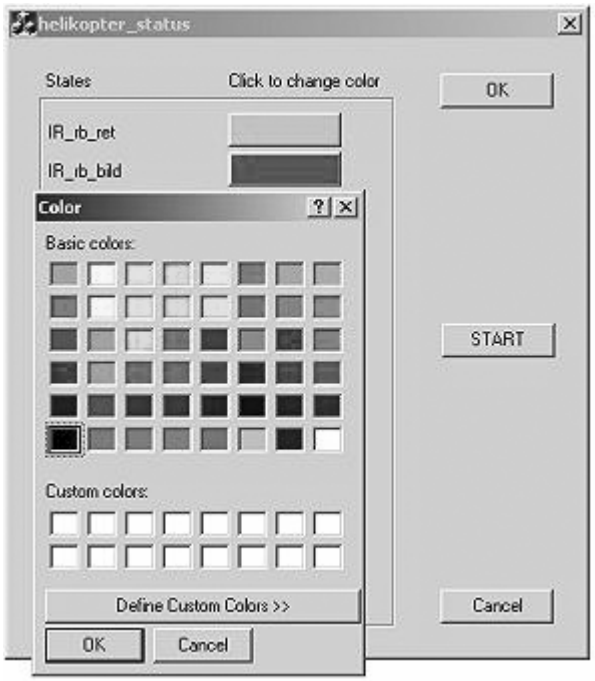
GUI:t består av dels gränssnitt för färginmatning, dels grafisk presentation av resultatet. I det här fallet blir det två separata problem.

3.4.1 Färginmatning:

Inmatnings-GUI:t skall tillhandahålla ett enkelt intuitivt verktyg för att välja vilken färg varje status skall ha. Två huvudalternativ finns: MFC [13] och LynX-modul. Arbetet som erfordras för varje alternativ kan anses vara lika varpå detta ej har någon betydelse för avgörandet vilket alternativ som skall väljas. Båda alternativen liknar varandra till utseende och funktionalitet och under följer en beskrivning av alternativen samt vad som skiljer de olika inmatningssätten ifrån varandra?

1. MFC

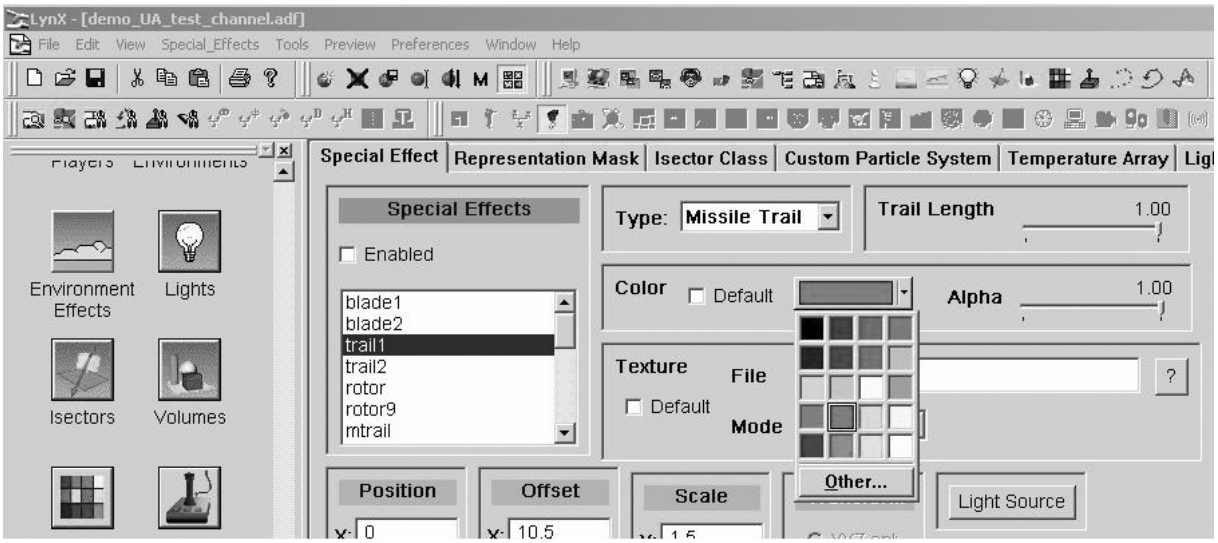
MFC är Microsofts egna klasser och kan anses vara en vedertagen standard för windowsprogrammering. Detta alternativ grundar sig i tanken att färginmatning sker i det huvudprogram som användaren kommer att använda i det färdigställda ”Terrängmodell Upptäckt”. På detta sätt blir det färdiga programmet mer fristående från Vega och LynX. Nackdelen med att använda MFC till detta ändamål är att fler klasser tillkommer i ”Terrängmodell Upptäckt” och ytterligare fönsterhantering erfordras. På figur 2.3 illustreras typisk färginmatning via MFC.



Figur 3.1 Färginmatning i MFC

2. LynX-modul

Lynx är ett grafiskt gränssnitt mot VEGA i vilket man kan konfigurera sina moduler och spara konfigurationen på fil som sedan kan anropas vid initilailseringen av VEGA och därmed konfigurera upp Vegas moduler. VEGA har ett C++ API. På figur 3.2 kan en del av det grafiska gränssnittet till LynX ses. På figuren ses hur en specialeffekt vid namn ”Missile Trail” konfigureras. Alternativet med en LynX-modul skulle på liknande sätt konstrueras vad det gäller färginmatning och ha en koppling till ett LynX-objekt (i detta fall helikopterobjektet). Skrivs modulen tillräckligt generell kan den användas till andra simuleringsmodeller. Sett ur ”Terrängmodell Upptäckt”-projektets synvinkel blir lösningen mindre flexibel eftersom användaren vid färginmatning måste använda LynX utöver det MFC-program som kommer att användas.



Figur 3.2 Färginmatning i LynX

Båda lösningarna har fördelar och nackdelar som beskrivits ovan. Eftersom färgval sker innan ett scenario simuleras är prestandaskillnad försumbar. Demonstrationsprogrammet ”Terrängmodell Upptäckt” kommer att byggas på MFC:s grafiska bibliotek tillsammans med att vissa avancerade parametrar editeras i LynX. För att bäst anpassa färginmatningen för det framtida program där det ska användas faller valet på MFC. Eftersom färgval av status kan förmodas vara något användaren av programmet vill kunna göra enkelt finns det uppenbara fördelar med att placera färgvalet i MFC-delen. Med det här valet av design skiljs den grafiska representationen från logik i ännu högre grad. Ändringar i den ena delen kan ske utan att påverka den andra. Detta tillåter helt oberoende omstrukturering av logikdelen och GUI:t om önskemål därom skulle finnas.

3.4.2 Resultatpresentation.

Till GUI:t räknas även statuspresentationen i form av spåret som ritas ut efter helikoptern under simuleringen.
Efter en inledande undersökning har följande två huvudalternativ framkommit:

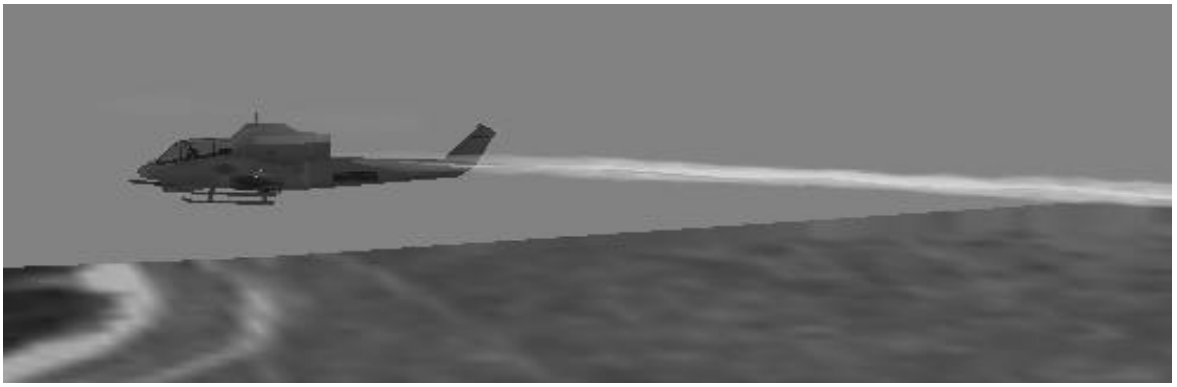
1. Inbyggd funktion, missile-trail
2. Rita linjer med hjälp av OpenGL [7] direkt i Vega.

Missile-trail är en specialeffekt som finns i Vega för att visualisera rök efter en missil, därav namnet. På detta spår kan färg, storlek, intensitet mm ändras. Första ansatsen var att använda missile-trail effekten p.g.a. den redan fanns och endast behövde konfigureras. Vid en närmare analys av funktionaliteten framkom att två hinder måste övervinnas för att missile-trail skulle kunna användas. Dels verkar livslängden på objektet vara tidsbegränsat, dels verkar det vara problem med att ha olika färg på olika etapper av spåret. Båda dessa hinder går eventuellt att övervinna med hjälp av listig uppdelning.

Det andra huvudalternativet är att rita linjer direkt till 3D-motorn. Att rita linjer direkt till 3D-motorn visade sig vara omständligt i Vega. För att åstadkomma detta måste ett anrop i 3D-motorns renderingsprocess deklareras vilket anropar en funktion med OpenGL-kod. I denna funktion kan text, linjer mm visualiseras genom att skriva OpenGL-kod. Fördelen med detta alternativ är att OpenGL är kraftfullt och större kontroll och flexibilitet uppnås.

Valet föll på att rita ut spåret med OpenGL av ovan nämnda skäl samt att visualiseringen blev tydligare med denna metod.

På följande två figurer kan visualiseringen av alternativen jämföras:



Figur 3.3 Spåret ritas ut med hjälp av plugin:en Missile-trail

Figur 3.4 Spåret ritas ut med hjälp av OpenGL

3.5 Resultat av analysen

1. Vad är huvuduppgiften?
Huvuduppgiften består av att designa och implementera en beräkningsmodul som är förberedd för nya krav och modifieringar.

2. Hur ska logikdelen designas?
Logikdelen skall följa ett objektorienterat tankesätt, vara flexibel, lättförstålig samt lätt att uppgradera. De fem viktigaste klasserna blir: Avdömare, Vega, Regelverk, Helikopter och Hotsystem.

3. Hur skall GUI-delen utformas?

Ett intuitivt gränssnitt skall utformas och detta skall göras med hjälp av MFC. I gränssnittet skall färger för olika status enkelt kunna konfigureras och simuleringen skall kunna startas. Varje status har en knapp med motsvarande respektive färg. För att ändra färg trycks statusens knapp ner och lämplig färg väljs ur en lista med färger.

Spårfärgen som representerar helikopterns status genereras med hjälp av OpenGL-funktioner under renderingen av bilden.

4. Vilka begränsningar finns och vilken prestanda kan förväntas krävas?

Ju fler hotsystem som finns med i simuleringen desto fler parametrar måste tas i beaktning när avdömningen skall ske. Uppdateringsfrekvensen av helikopterns status blir någorlunda linjärt beroende av antalet hotsystem. Antalet hotsystem som kan användas i en simulering blir därför beroende av kravet av lägsta uppdateringsfrekvens. Den i sin tur beror på en rad faktorer såsom beräkningsmodulens effektivitet, grafikmodellernas storlek, beräkningskapaciteten i den dator där simuleringen körs. Den lägsta uppdateringsfrekvens som behövs för att ej mista känslan av realtidssimulering är subjektivt beroende på användaren men 25 uppdateringar per sekund har visat sig vara tillräckligt.

5. Vilka antaganden, uppdelningar och förenklingar kan/bör göras?

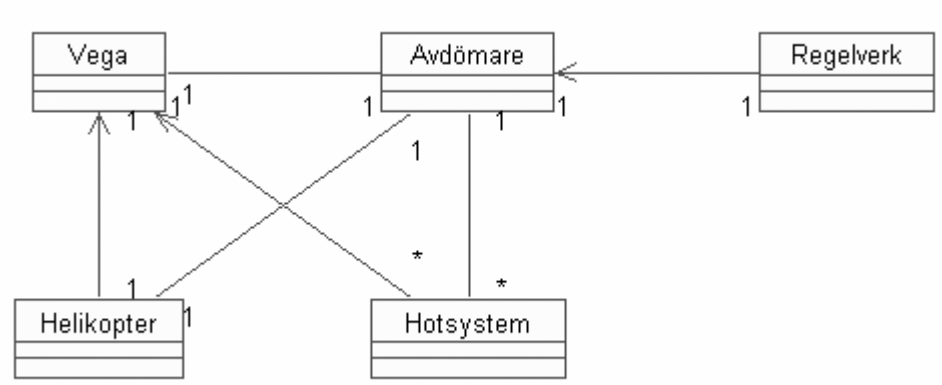
IR-robot av bildalstrande typ och av retikel typ behandlas som två separata system. Vid siktlinjesberäkningar utförs beräkningar mot en punkt och inte mot en volym.

4 Design

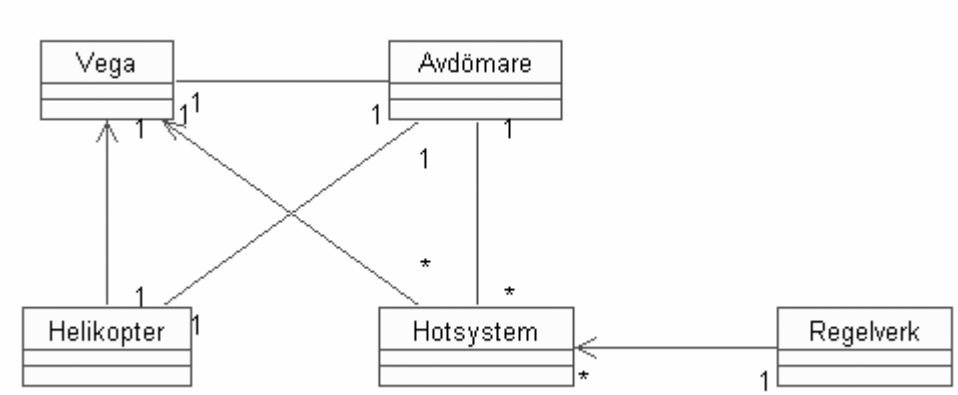
Analyskapitlet kom fram till att fem huvudklasser behövdes, dock framgår det ej i analyskapitlet hur dessa klasser skall sammankopplas. Följande klasser behövs:

- Hotsystem: Hanterar intern status, interna systemtider, position.
- Helikopter: Hanterar intern status, position, antal facklor och remsor.
- Avdömningsklass: Avdömer hotsystemets och helikopterns status.
- Regelverk: Tillhandahåller de regler som avdömningsklassen behöver för att bestämma respektive status.
- Vegaklass: Hanterar kopplingen mellan 3D-motorn och helikopter och hotsystem.

Två förslag på möjlig design av sammankopplingen av huvudklasserna har framförallt tagits i beaktning. Förslag 1 (figur 4.1) visar en design där avdömarklassen kontrollerar sammankopplingen mellan regelverk och hotsystem för att slutligen kunna avdöma status för helikoptern. Förslag 2 (figur 4.2) grundar sig i tanken på att de regler som hotsystemen behöver för att kunna avgöra hur de förhåller sig till helikoptern är så nära förknippat med hotsystemen att de är direkt kopplade till hotsystemen. Ytterligare en möjlighet är att slå ihop hotsystem och regelverk på grund av att kopplingen mellan dem är så stark. Denna hypotetiska klass hade då bildat en basklass och respektive hotsystemtyp hade ärvt grundläggande funktionalitet från denna.



Figur 4.1 Schematisk bild av designförslag 1. Avdömarklassen har en central roll



Figur 4.2 Schematisk bild av designförslag 2. Hotsystemklassen har en central roll

Det som slutligen lagt grunden för detta är strävan att nå en så objektorienterad lösning som möjligt. Därför har förslag 1 valts. Avdömarobjektet hämtar de parametrar som behövs ur ett hotsystemobjekt och skickar in parametrarna i motsvarande regelobjekt som returnerar ett svar. Sett ur ett objektorienterat synsätt är det avdömarens uppgift att knyta sig an regelverk eftersom det är avdömaren som använder sig av regelverket. Ytterligare en anledning till att inte knyta regler till hotsystem är att lösningen blir mer flexibel. Om det i framtida regelverk framkommer nya förutsättningar för att uppnå specifik status skulle väldigt mycket kod behöva justeras och/eller uppdateras.

5 Implementering

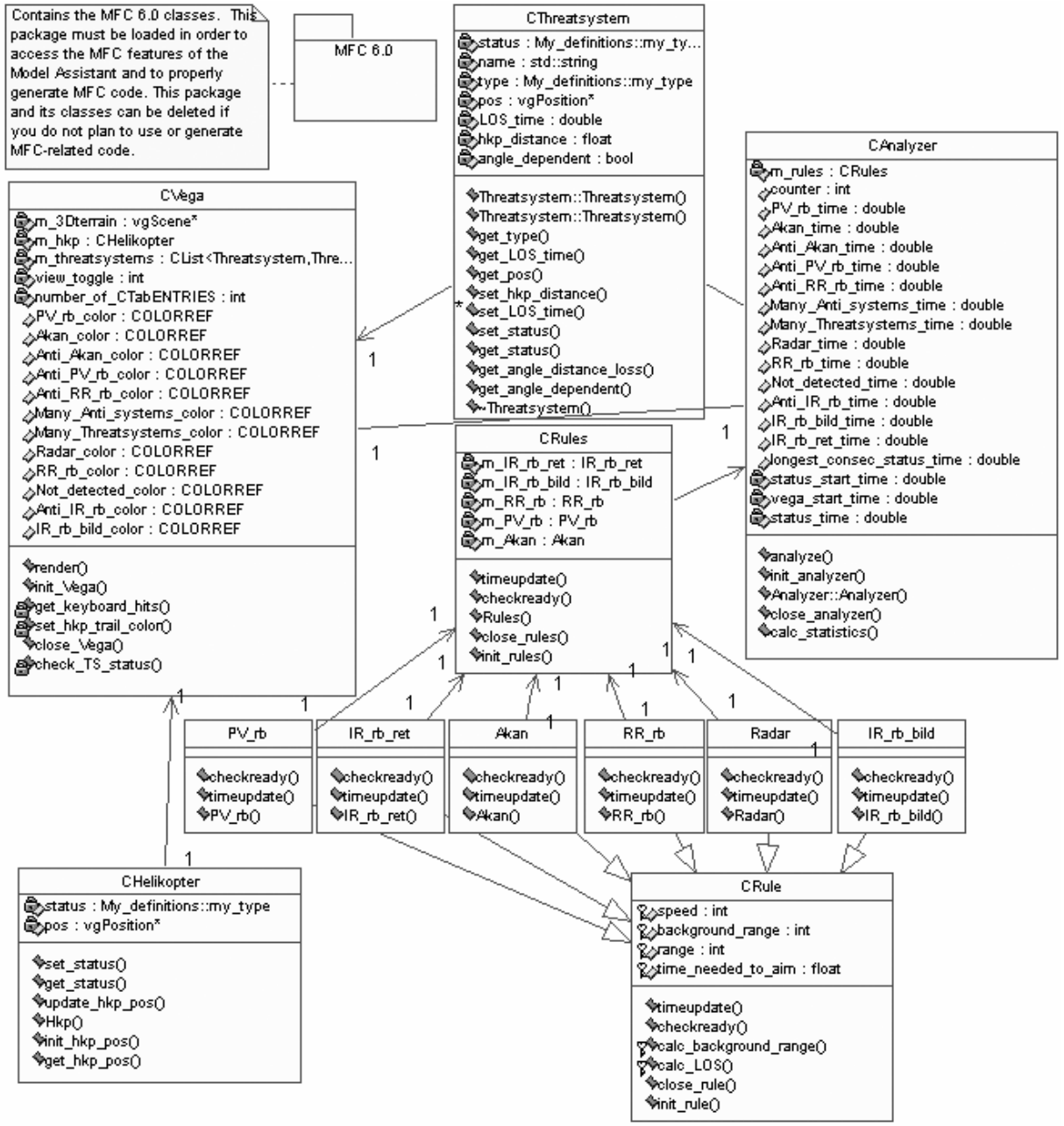
Implementeringen är uppdelad i logikdel respektive GUI-del.

Resultatet av examensarbetet är en implementering av dels den statusberäknande delen och dels delen som presenterar vilken status helikoptern har i simuleringsprogramvaran ”Terrängmodell Upptäckt”.

5.1 Implementering av logikdel

Implementeringen av logikklasserna är en tidskrävande process och här följer bara en kort beskrivning av processen. Den börjar med generering av klasskeletten från Rational Rose till ett ”command line project” i Visual C++. Efter detta började arbetet med att implementera metoderna. Även om huvuddelen av designen var gjord tidigare gjordes flera återföringar till Rational Rose för att göra mindre förändringar i designen. Att överföra kod tillbaka till UML-klasser (se figur 5.1) och göra förändringar där har fördelen att designförändringar blir översiktliga och oförutsedda bieffekter kan undvikas. De allra flesta modifieringarna uträttas dock lättast genom att direkt ändra i koden. Eftersom engelska är det enda vedertagna språket i datorsammanhang har alla klasser, metoder, variabler fått engelska namn.

Tidigare är regelverket bara refererat till som en enda klass. I implementeringen visade sig dock fördelaktigt att inför en basklass (Rule) (se figur 5.1). Alla specifika hotsystem har sedan en regelklass som ärver funktionalitet från ”Rule”. De övriga systemet kopplas bara mot regelverket (Rules-klassen) som tillhandahåller kopplingen från ”hela” regelverket.



Figur 5.1 Klasser, metoder och medlemsvariabler skapade med Rational Rose

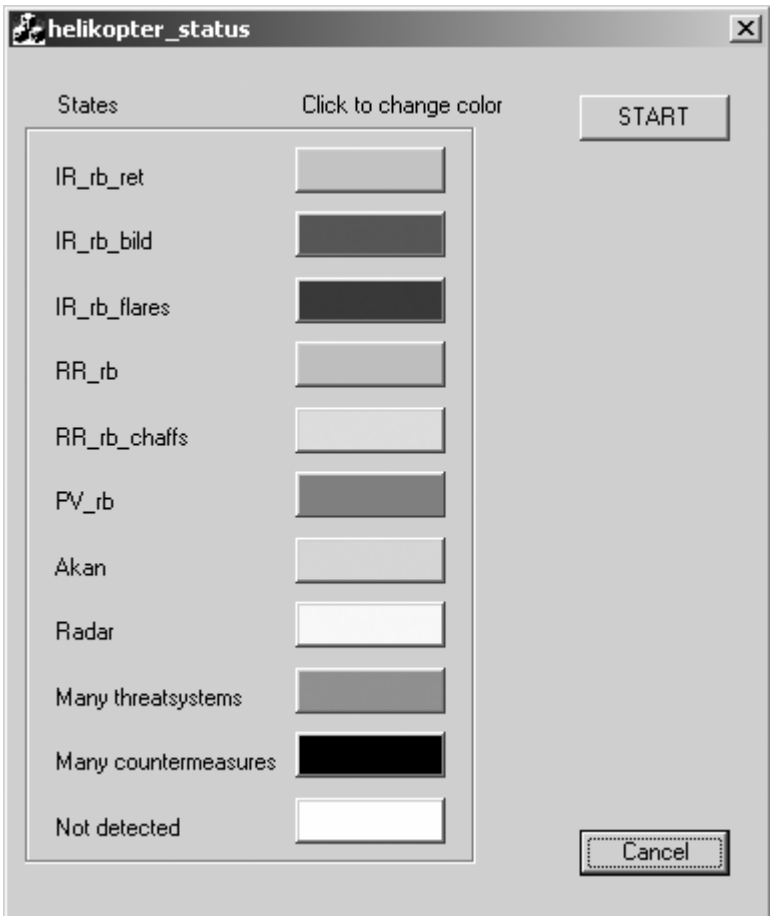
5.2 Implementering av GUI-del

Ett ”Dialogbased” projekt skapades i Visual C++. För att åstadkomma knappar med färg på användes klassen CcolorButton [12]. En CcolorDialog är knuten till varje tryck på färgknapp. Vid implementeringen av spårgenereringen var en studie i Vegas renderingsuppbyggnad nödvändig. Detta p.g.a. Vegas sätt att exekvera OpenGL-kod.

6 Resultat

6.1 Hur modulen fungerar

Under följer en beskrivning av hur modulen fungerar vid exekvering. När programvaran startas visas dialogrutan som finns på figur 6.1.



Figur 6.1 Startdialogruta till modulen

I detta dialogfönster görs eventuella färgval till statusen genom att trycka på respektive knapp och välja färg. För att starta analysen av scenariot trycker man på startknappen.

Programvaran börjar med att starta Vega och sätta upp scenariot. Parametrarna till scenariot som terrängavsnitt, helikopterbana, hotsystem läses in från en fil som konfigurerats i LynX.

När simuleringen körs (analysförfarandet) bestäms helikopterns status med ett visst tidsintervall. Avdömningen sker i två steg. Först anropas regelverket för varje hotsystem och dess interna status bestäms. Detta sker med hjälp av siktlinjesberäkningar (Vega isectors) samt tidsmätningar. I steg två sker avdömningen av helikopterns status beroende av en prioritetsordning hos de aktuella hotsystemens status.

Här beskrivs hur programmet fungerar på klassnivå. Ett varv i mainloopen ser i pseoudokod ut så här:

```
Mainloop
{
        anropa analyze;
        anropa render;
}
Analyze
{
        Gå igenom hotsystemlistan och uppdatera hotsystemets interna status;
        Gå igenom hotsystemlistan för att få reda på vilket status helikoptern ska ha;
}
Render
{
        Konfigurera vyer;
        Uppdatera helikopterns position;
        Sätt färgen på helikopterspåret;
        Rendera scenen;
}
```

Här följer en liten närmare beskrivning av vad som sker i avdömarfunktionen (analyze). När hotsystemlistan itereras bestäms hotsystemets status. Status för ett hotsystem kan vara: ej aktiv, aktiv, bekämpningsbar. Bekämpningsbar betyder i det här fallet att motmedel kan användas mot hotsystemet och gäller bara vissa hotsystem.

Exempel på statusförändring hos en radarrobotstation
Radarstationens status: ”ej aktiv”.

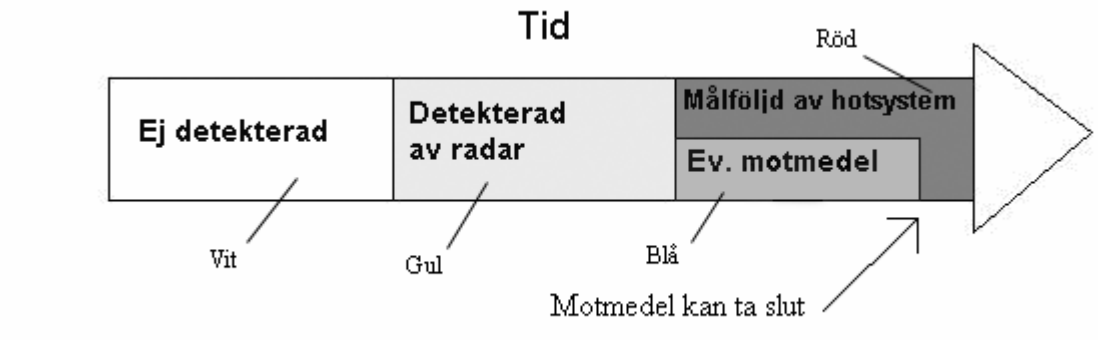
1. Radarstation invisar helikopter och radarrobotstationen uppfyller följande krav (kraven är figurerade men typsenliga):
    1. Helikopter inom 800 meters avstånd.
    2. Fri sikt till helikoptern samt fri sikt 15 meter i riktningen mot bakgrunden till helikoptern i 2 sekunder.
    3. Är detta uppfyllt får radarstationen statusen ”aktiv” annars fortsätter den att vara ”ej aktiv”
2. Om radarstationen är ”aktiv” och den tid som är angiven för robotskottsvarningstid är uppnådd byts radarstationens status till ”bekämpningsbar” förutsatt att helikoptern är bestyckad med remsor som är motvapnet mot radarrobot.
3. Om statusen är bekämpningsbar och remsorna tar slut återgår radarrobotsstationens status till ”aktiv”.

Slut på exemplet. De andra hotsystemen fungerar på motsvarande sätt men med sina specifika krav. Observera att motmedel inte kan användas mot alla hotsystem och att dessa bara har två olika status: ”ej aktiv” respektive ”aktiv”.

Efter att hotsystemlistan är itererad i avdömarfunktionen, bestäms helikopterns status från en prioritetslista: Statusen har följande prioritet i fallande ordning:

1. Motmedel används mot multipla hotsystem
2. Motmedel används mot ett hotsystem
3. Multipla hotsystem är aktiva
4. Ett hotsystem är aktivt
5. Radar detekterar helikopter
6. Ej detekterad

Ett normalt scenario där en eller flera radarstationer finns samt ett hotsystem kan få följande status sett över tidsaxel (se figur 6.2).



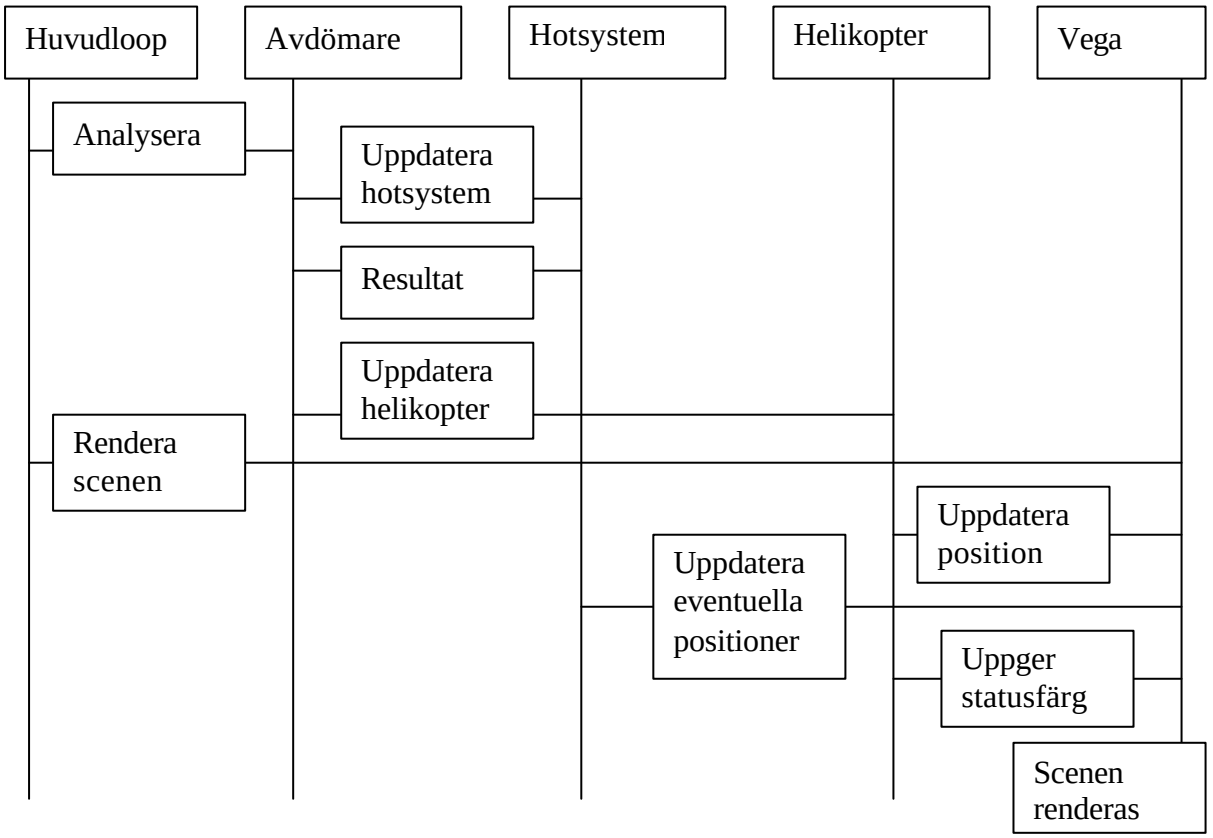
Figur 6.2 Tidsaxel för helikopters status

Nästa steg i huvudloopen är att anropa en ”render-funktion” i klassen Vega.

I render-funktionen konfigureras betraktningsvyerna beroende på användarens val. Användaren kan när som helst under simuleringen byta betraktningsvy genom en knapptryckning på tangentbordet. Sedan anropas de funktioner som uppdaterar

helikopterns och eventuella hotsystems positioner vilket sker via Vegas ”motion-model” som är inlästa från LynX. En ”motion-model” i Vega är en del som beskriver ett Vegaobjekts banrörelse och fart i 3D-terrängen. Det näst sista som görs i render-funktionen är ett anrop till en funktion som lägger till helikopterns koordinater samt statusfärg i en lista. Sist görs ett Vega anrop för att rendera scenen. Det är sedan Vega som i OpenGL-funktionen får tillgång till listan med helikopterns position och statusfärg och ritar ut den i det sista skedet av renderingen av bilden.

På figur 6.3 finns ett sekvensdiagram som schematiskt beskriver förloppet ett varv i huvudloopen. Den beskriver vilka kopplingar (eller anrop) som görs mellan huvudklasserna. Den skall tolkas som att tidsaxeln pekar neråt, d.v.s. först anropar huvudloopen avdömmarfunktionen, därefter avdömmarfunktionen hotsystemen och så vidare. Allra sist resulterar loopen i att scenen renderas (och ett nytt varv i huvudloopen kan påbörjas).



Figur 6.3 Samarbetes eller sekvensdiagram av ett varv i huvudloopen

6.2 Hur väl mål och krav uppfylldes från kravspecifikationen

Målet bestod i att:
Designa och implementera modellen för vilken status helikoptern har i förhållande till spanings- och vapensystemen samt att presentera detta som ett färgkodat spår bakom helikoptern i 3D visualiseringen.

Huvudmålet är uppnått. Mindre modifieringar av kravspecifikationen har varit nödvändigt att göra i efterhand, därför finns det vissa delar av implementeringen som ej överensstämmer med kravspecifikationen. Modifieringarna i kravspecifikationen har

antingen skett på begäran av handledare eller i samförstånd med handledare. Exempel på detta är utvidgandet med ytterligare en status genom separerandet av IR-robot retikel och IR-robot bildalstrande.

Vad det gäller de fyra möjliga nivåerna av implementeringen så kan A-C, förutom vinkelberoendet i B, anses ha implementerats (dock under aningen längre tid än tio veckor).

7 Diskussion

7.1 Framtida användning av modulen

7.1.1 Praktisk användning

Detta examensarbete initierades med avseendet att den resulterande beräkningsmodulen skulle kunna praktiskt användas i simuleringsmodellen ”Terrängmodell Upptäckt”. I det färdiga projektet ”Terrängmodell Upptäckt” används denna modell om än i modifierad form. Den skapade modulen lade grunden för det scenarioanlyseringsverktyg som blev en av de centrala delarna i det färdiga projektet ”Terrängmodell Upptäckt”.

7.1.2 Vilka modifieringar behövde göras

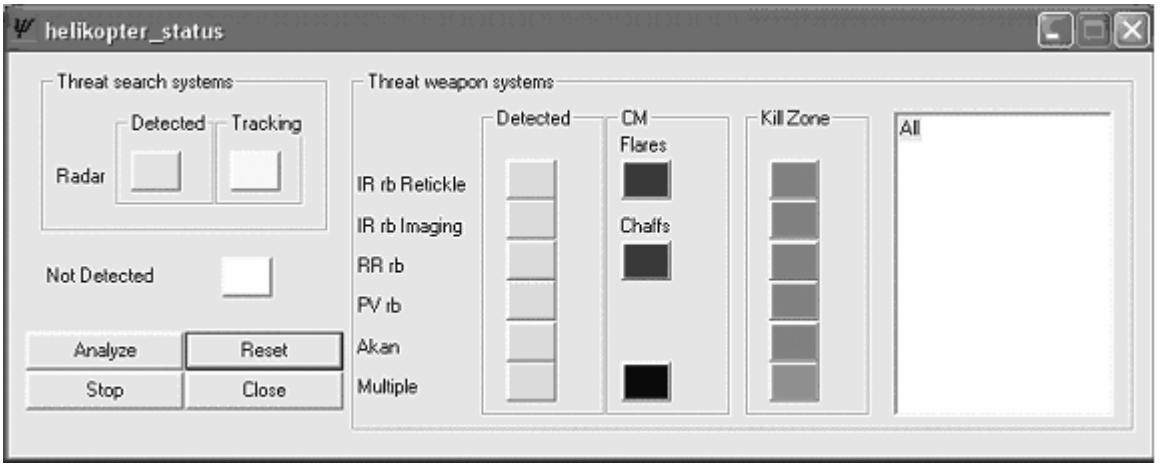
De modifieringar som gjordes för att användas i simuleringsprogramvaran ”Terrängmodell Upptäckt” berodde på nyuppkomna krav och önskemål på modellen. Dessa önskemål och krav tillkom under slutfasen av projektet, flera månader efter den praktiska delen av examensarbetet slutförts.

I den färdiga programvaran tillkom nya status. Det stadium där hotsystem ”siktar in sig” blev till ytterligare ett statusläge. Från att tidigare gått direkt från ”ej detekterad” till ”målföljd” infördes stadiet ”detekterad” som tidigare bara hanterats internt i beräkningsmodellen. Förloppet blev alltså ej detekterad -> detekterad -> målföljd i det slutliga projektet.

Behov uppstod av att skilja på spaningssystem (radarstationer) och vapensystem. Dessutom tillkom krav på statistikinsamling i form av tiddata för varje status i respektive hotsystem samt för helikopter.

Trots dessa modifieringar visade sig grundmodellen med de fem huvudklasserna vara en tillräckligt flexibel modell och de nya modifieringarna kunde lätt appliceras utan att grundstommen i beräkningsmodellen behövde förändras. Vad gäller GUI-delen behövde bara mindre ändringar göras i färginmatningsfönstret p.g.a. de status som nytillkommit. Grundprincipen förblev dock intakt. Spårgenereringen behövde ej modifieras alls utan förblev i sin ursprungliga form.

Färgvalsfönstret därifrån simuleringen initieras kom att se ut enligt figur 7.1:



Figur 7.1 Färginmatning i den slutgiltiga användningen av modulen

I appendix B finns ett scenario ur det färdiga ”Terrängmodell Upptäckt” där analysfunktion används.

7.1.3 Möjliga förbättringar

En möjlig förbättring för att förbättra realismen är att konstruera en egen volymkollisionsdetektering varpå ”riktiga” siktlinjesberäkningar skulle kunna utföras.

7.1.4 Ytterligare användning?

Eftersom denna beräkningsmodul är så problemspecifik är det ej troligt att denna lösning är applicerbar på andra program. Inte heller andra delar av kod eller grafiskt gränssnitt kommer att kunna utnyttjas i någon större utsträckning. Enda möjliga återanvändning av beräkningsmodul och GUI är om en liknande simuleringsprogramvara för VMS skall konstrueras för någon annan typ av farkost (t ex, pansarvagn, flygplan).

7.2 Vad som kunde gjorts bättre

Sett i efterhand utfördes examensarbetet väl, med möjligen undantag av tidfördelningen. Implementeringsarbetet prioriterades varpå rapportskrivning ej kunde färdigställas under de tio veckor som examensarbetet var planerat att behöva. I övrigt är kraven i kravspecifikationen uppfyllda och resultatet i form av mjukvara har visat sig användbar.

8 Referenslista

De referenser som ej är refererade till i rapporten har fungerat som hjälpmedel eller inspirationskälla under arbetets gång.

8.1 Handlingar

[1] Claes Johansson, *”Delrapport VMS Hkp (HKP99352S)” efter år 2000*
Helikopterflottiljens beteckning 21 120:10061, Datum 2000-01-17

[2] Claes Johansson, *Delrapport HKP99352S VMS för helikopter år 2001*
Helikopterflottiljens beteckning 21 120:10955, Datum 2001-12-13

8.2 Websidor

[3] Microsoft. *Microsoft Visual C++*.
http://msdn.microsoft.com/vstudio
(tillg: 2003-04-06)

[4] Multigen-Paradigm.*Vega.*
http://www.multigen-paradigm.com
(tillg: 2003-04-06)

[5] Rational Software. *Rational Rose*.
www.rational.com
(tillg: 2003-04-06)

[6] OMG. *UML.*
http://www.uml.org
(tillg: 2003-04-06)

[7] *OpenGL.*
www.opengl.org
(tillg: 2003-04-06)

[8] Multigen-Paradigm. *Vega forum.*
http://www.multigen-paradigm.com/cgi-bin/Ultimate.cgi
(tillg: 2003-04-06)

[9] *OpenGL information*
http://nehe.gamedev.net/
(tillg: 2002-06-20)

[10] *C++ kod exempel*.
http://www.programmersheaven.com/zone3/mh22.htm
(tillg: 2003-04-06)

[11] SIG. *OpenGL performer*.
http://www.sgi.com/software/performer/

(tillg: 2003-04-06)

[12] Paul J. Weiss. *ColorButton kod/exempel*.
http://www.codeproject.com/miscctrl/colorcontols.asp
(tillg: 2003-04-06)

[13] Microsoft. *MFC*
http://msdn.microsoft.com/library/devprods/ vs6/visualc/vcmfc/mfchm.htm
(tillg: 2003-04-06)

[14] *RGB värden*.
http://www.htmlhelp.com/cgi-bin/color.cgi?rgb=FFFFFF
(tillg: 2003-04-06)

8.3 Böcker

[15] Gamma, Erich. Design Patterns – elements of reusable object-oriented software. Addison-Wesley, 2000

[16] Delaura, Mark A. Game Programming gems. Charles River Media, *2000.*

[17] Skansholm, Jan. C++ direct*. Studentlitteratur, 1996*

[18] Martin Fowler. UML distilled, second edition*. Addison-Wesley, 2000*

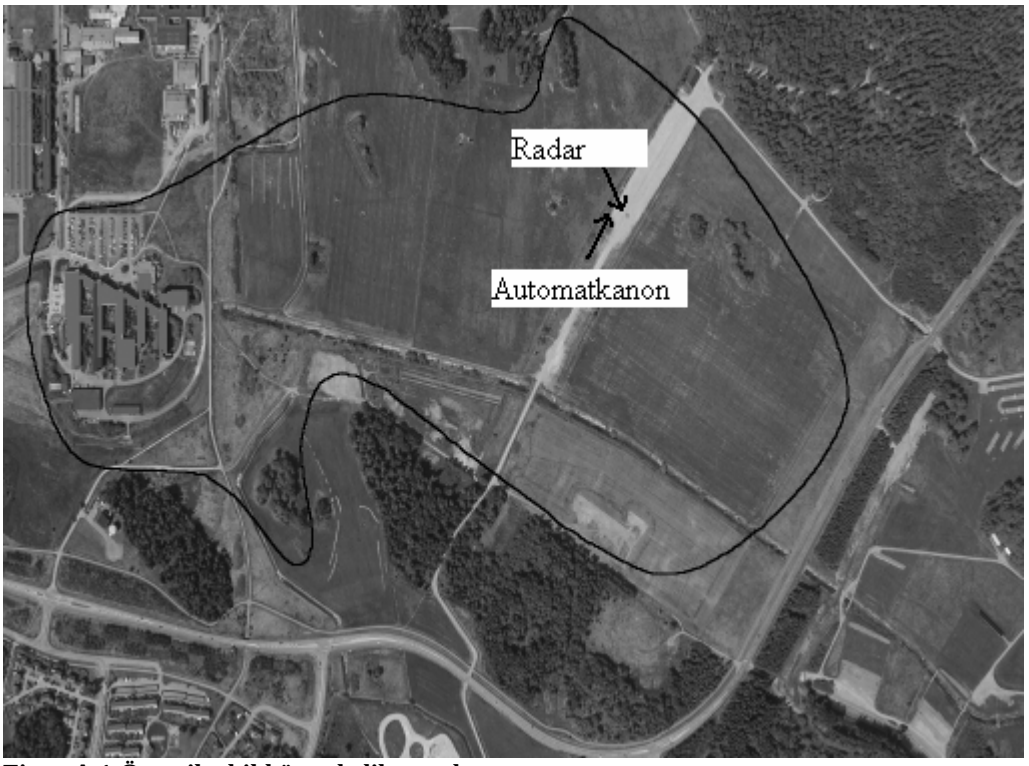
8.4 Manualer

[19] Multigen Paradigm. VEGA – Programmers guide for Microsoft Windows. *Augusti 2000.*

[20] Rational Software Corporation. Using Rose Visual C++ - Rational Rose 2000e. *2000.*

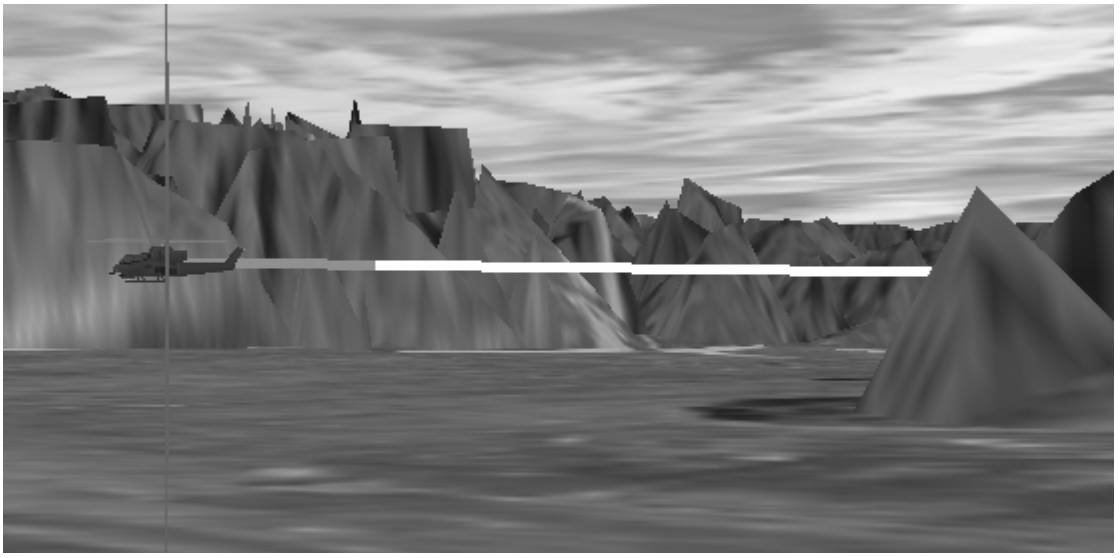
Appendix A

Exempel på scenario där beräkningsmodulen används för att analysera en helikopterbana.
I detta scenario är en spaningsradar, en automatkanon utplacerade bredvid varandra i landskapet. Helikoptern flyger, på ca 10 meters höjd, i en bana runt hela terrängen. På figur A.1 visas helikopterbanan som ska flygas med en svart linje. Både radarstationen och automatkanonen är utplacerade relativt öppet. Helikoptern flyger dels över öppna ytor, dels i skydd av hus och skog. På några ställen flyger helikoptern även längs skogsridåer för att därigenom undvika att bli upptäckt.



Figur A.1 Översiktsbild över helikopterbanan

När simuleringen startas flyger helikoptern längs helikopterbanan och avdömningen av helikopterns status sker i realtid. På figur A.2 visas en skärmdump från simuleringen utifrån radarstationens vy. Helikoptern har precis kommit ut bakom ett skogsparti och blivit synlig för radarstationen. Den vita färgen representerar här att helikoptern ej är detekterad. Efter någon/några sekunder har dock radarstationen identifierat helikoptern och helikopterns status ändrar färg till gult som i det här fallet representerar att helikoptern är detekterad av radarstationen.



Figur A.2 Radarvy från simulering

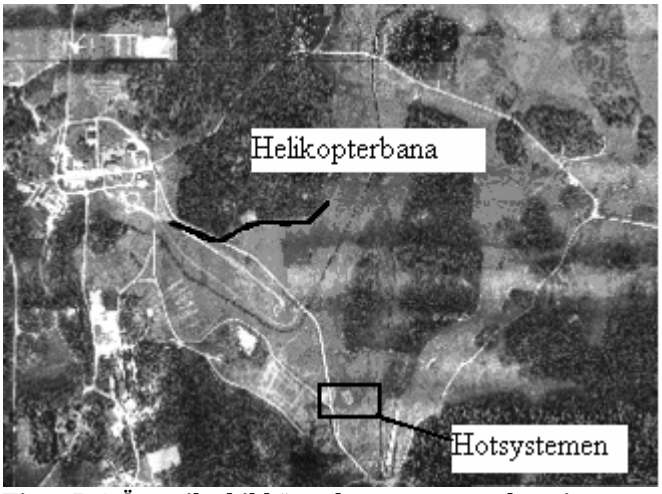
På figur A.3 visas en översiktsbild efter att hela simuleringen är slutförd och resultatet kan analyseras. Vitt spår = ej detekterad, grått spår = detekterad av radar, svart spår = bekämpningsbar av automatkanonen. På ställen där helikoptern har flugit nära skogsridåer har radarstationen haft svårt att detektera helikoptern på grund av att helikoptern smultit in i bakgrunden. Helikoptern har också klarat sig ifrån upptäckt och bekämpning väl på ställen där den varit skymd. På vissa delsträckor har dock helikoptern flugit på öppna ställen och helikoptern har då detekterats av radarstationen och på vissa sträckor varit bekämpningsbar av automatkanonen.

Figur A.3 Översiktsbild med resultatet från simuleringen

Appendix B

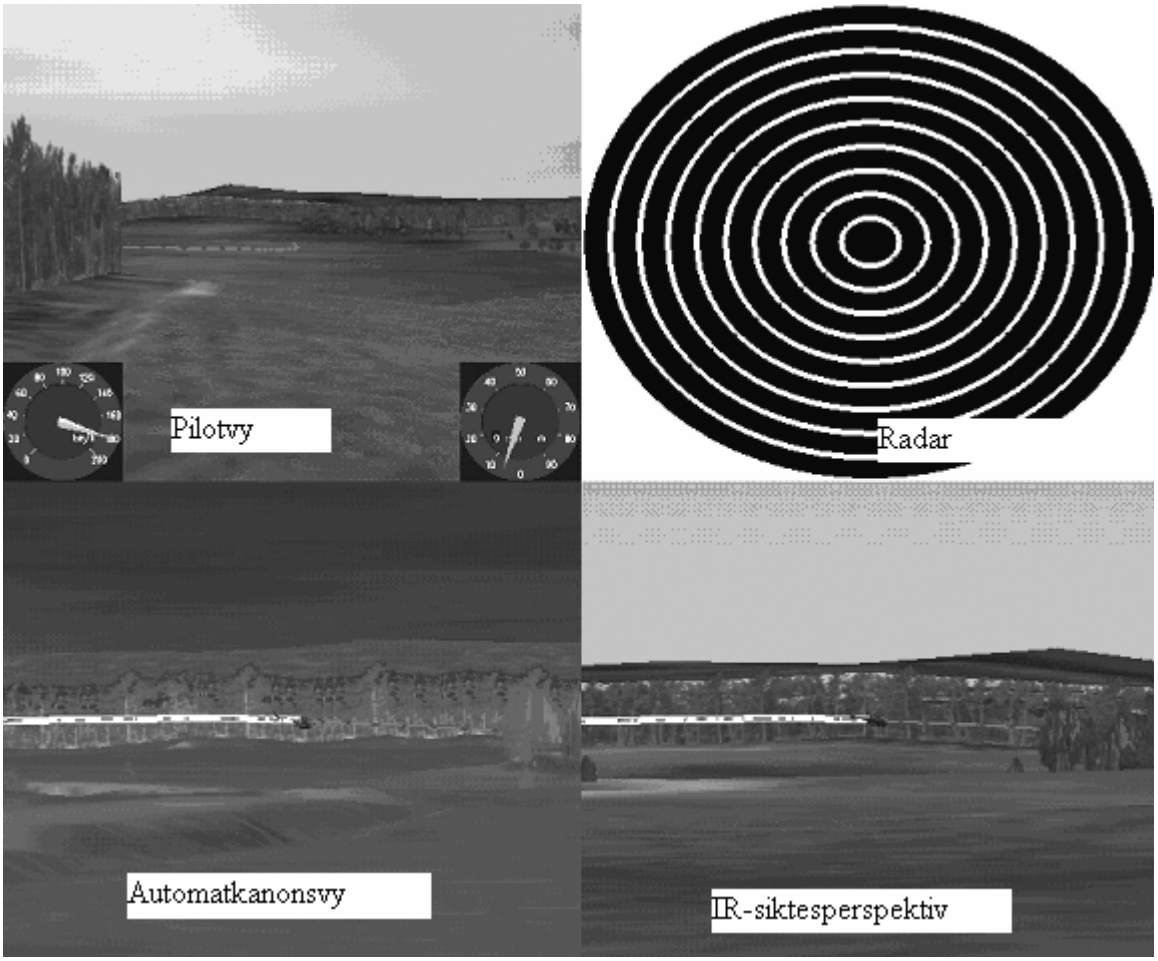
Exempel på scenario där beräkningsmodulen används i det färdigställda ”Terrängmodell Upptäckt”.

I detta scenario är en spaningsradar, en automatkanon och en IR-robot med retikelmålsökare utplacerade bredvid varandra i landskapet. Helikoptern flyger, på ca 5 meters höjd, i en bana nära en skogsridå. Se figur B.1.



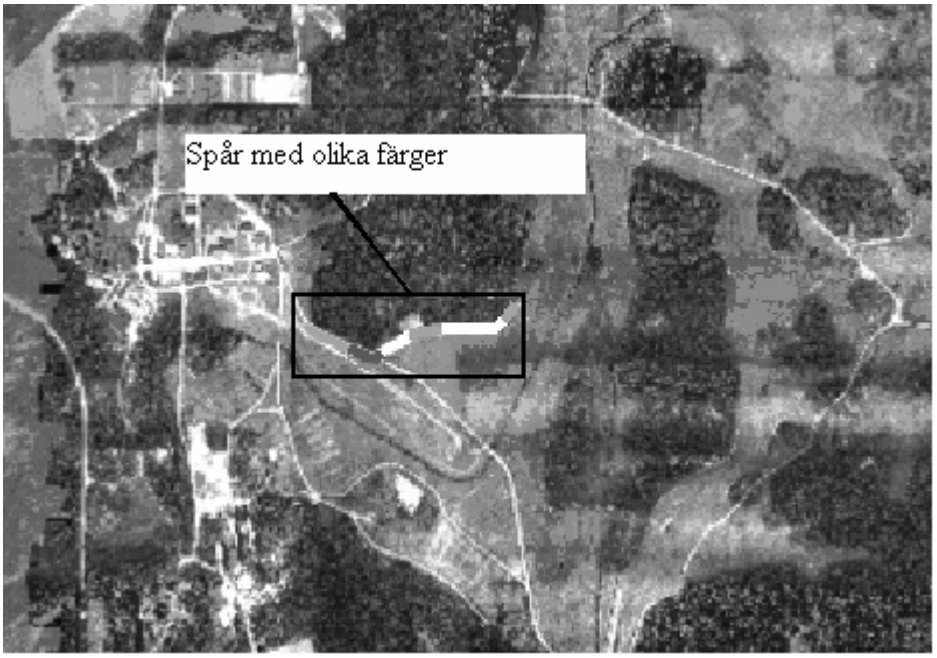
Figur B.1 Översiktsbild över hotsystemens placering samt helikopterns färdväg

Under själva simuleringen kan en skärmdump från programmet se ut enligt figur B.2.



Figur B.2 Pilotvy, radar,automatkanonsvy, IR-siktesperspektiv

Spaningsradarn är i detta scenario modellerad med en avståndslucka på 30 meter. Under en stor del av helikopterns färdväg hamnar helikoptern och skogsridån i samma avståndslucka, vilket medför att helikoptern är osynlig på radarn trots att den är synlig i de visuella och IR-siktesbilderna, se figur B.3. Statistik från körningen finns i tabell B.1.



Figur B.3 Resultat av analysen. Vitt spår = helikoptern är ej detekterad av något hotsystem

Tabell B.1 Statistik från scenario

Statistics:			%	s
	Total Runtime:		100.0	12.3
	Not Detected:		16.0	2.0
	Longest Consecutive Status: (Akan):		18.9	2.3
Radar:				
	Detected:		31.9	3.9
	Tracking:		16.0	2.0
	Radar1:			
		Detected:	31.9	3.9
		Tracking:	16.0	2.0
Akan:				
	Detected:		32.9	4.0
	Kill Zone:		34.5	4.2
	Akan1:			
		Detected:	32.9	4.0
		Kill Zone:	34.5	4.2
IR-rb ret:				
	Detected:		74.3	9.1
	CM:		0.0	0.0
	Kill Zone:		0.0	0.0
	IrRbRet1:			
		Detected:	74.3	9.1
		CM:	0.0	0.0
		Kill Zone:	0.0	0.0
Multiple threats:				
	Detected:		25.1	3.1
	CM:		0.0	0.0
	Kill Zone:		0.0	0.0

Appendix C

Ordförklaringar

3D-motor	Mjukvara för att utföra 3D-beräkningar (bland annat förflyttning, vridning, skalning) av 3D-modeller och rendera en bild av 3D-världen
Bildalstrande	Modern typ av målsökare
C++	Ett av de världsledande programmeringsspråken
COTS	Förkortning av “Commercial Off The Shelf” = hyllvara. Kommersiell mjukvara som används som bottenplatta för vidare mjukvarutveckling
Exekverbar kod	Körbar programmeringskod
GUI	Förkortning av ”Graphical User Interface”, grafiskt användargränssnitt.
Pseudokod	Text som efterliknar mjukvarukod men skrivs med vanlig förklarande text
Retikel	Karakteristisk del i äldre typ av målsökare
Siktlinjesberäkning	En beräkning i 3D-världen som utförs för att ta reda på var och vad som stöts på i en viss riktning utifrån en viss punkt i 3D-världen.
Skärmdump	En bild som är en exakt kopia av vad som visades på bildskärmen vid en specifik tidpunkt
Systemtid	En tidsuppmätning av simuleringsprogramvaran.
UML	Förkortning av ”Unified Modelling Language”. UML är ett standardiserat språk för att designa/modulera mjukvara