FOI-R--0846--SE
April 2003
ISSN 1650-1942
Metodrapport

Erik Caidahl, Mats Olsson

Marschmodell för förband i ramverket FLAMES

Ledningssystem
581 11 Linköping

TOTALFÖRSVARETS FORSKNINGSINSTITUT

Ledningssystem
Box 1165
581 11 Linköping

FOI-R--0846--SE

April 2003

ISSN 1650-1942

Metodrapport

Erik Caidahl, Mats Olsson

Marschmodell för förband i ramverket FLAMES

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets Forskningsinstitut - FOI Ledningssystem Box 1165 581 11 Linköping	FOI-R--0846--SE	Metodrapport
	Forskningsområde 4. Spaning och ledning	
	Månad, år April 2003	**Projektnummer** E 7031
	Verksamhetsgren 5. Uppdragsfinansierad verksamhet	
	Delområde 41 Ledning med samband och telekom och IT-system	

Författare/redaktör	Projektledare
Erik Caidahl Mats Olsson	Per Svensson
	Godkänd av
	Uppdragsgivare/kundbeteckning FM
	Tekniskt och/eller vetenskapligt ansvarig Per Svensson

Rapportens titel

Marschmodell för förband i ramverket FLAMES

Sammanfattning (högst 200 ord)

Rapporten avser ett examensarbete utfört i projektet ”Informationsfusion i det nya försvarets ledningssystem” på FOI. Syftet med examensarbetet är att utveckla en modell för hur ett fordonsförband beter sig under marsch. Det ska även skapas en prototyp för detta anpassad till simuleringsramverket FLAMES.

Förbandens förflyttning ska vara baserad på doktriner, d.v.s. regelverk för hur ett förband uppträder. Resultatet av examensarbetet ska sedan användas för att testa de fusionsalgoritmer vars utformning utgör huvuddelen av Infofusionsprojektet. Förbandens beteenden ska också beskrivas i FLAMES och utnyttja ramverkets nya stöd för terrängdata. Denna rapport beskriver de beteendemönster som modellerats och hur dessa fungerar i ramverket FLAMES.

Nyckelord

simuleringsramverk, scenariosimulering, terrängmodell, doktrin

Övriga bibliografiska uppgifter	Språk Svenska
ISSN 1650-1942	**Antal sidor:** 51 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Issuing organization	Report number, ISRN	Report type
FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	FOI-R--0846--SE	Methodology report
	Programme Areas 4. C4ISR	
	Month year April 2003	**Project no.** E 7031
	General Research Areas 5. Commissioned Research	
	Subcategories 41 C4I	

Author/s (editor/s)	Project manager
Erik Caidahl Mats Olsson	Per Svensson
	Approved by
	Sponsoring agency FM
	Scientifically and technically responsible Per Svensson

Report title (In translation)
Marching model for military units in the FLAMES framework

Abstract (not more than 200 words)
This report describes a masters thesis within the "Information fusion in the command system of the new defence" project at FOI. The purpose of this work is to develop a model for the behaviour of mechanized military units while marching. A prototype for this behaviour should also be developed for use with the simulation framework FLAMES. The movements the military units perform should be based on doctrine, which are sets of rules for a unit's behaviour. The resulting prototype will be used for testing and evaluation of the fusion algorithms whose development is the main part of the Infofusion project. The behaviour of the military units should also be described in FLAMES and use the new terrain module included in the latest version of this program. This report describes the behaviour patterns which have been modelled and their implementation in FLAMES.

Keywords
simulation framework, scenario simulation, terrain model, doctrine

Further bibliographic information	Language Swedish

ISSN 1650-1942	Pages 51 p.
	Price acc. to pricelist

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Att på ett korrekt sätt kunna sammanställa och klassificera information ur stora informationsflöden under osäkerhet är ett område som alltid varit problematiskt, inte minst i militära sammanhang. I det nya försvaret, där information och underrättelser kommer att spela en allt större roll är det synnerligen viktigt att utveckla metodik för att ge stöd åt mänskliga beslutsfattare. För att undvika en överbelastning av dessa beslutsfattare är det viktigt att sortera, filtrera och tolka innebörden av den insamlade informationen. Mot denna bakgrund har forskningsområdet informationsfusion vuxit fram. Det handlar om att bearbeta stora informationsmängder som kan vara svåra att överblicka, ofullständiga eller innehålla felaktigheter. Informationen kan också komma från flera olika källor med varierande pålitlighet. Materialet ska sedan förvandlas till något som en beslutsfattare kan använda direkt i sin verksamhet. Hur man ska styra sina sensorer och informationsinsamlingsresurser för att på bästa möjliga sätt samla in denna information ligger också inom detta område. De algoritmer som utarbetas för informationsfusion behöver kunna testas i en realistisk miljö. För närvarande handlar detta till stor del om simuleringar och att bygga modeller av de områden där informationsfusion kommer att användas. Ett sådant område är detektion och klassificering av fientliga förband i stridssituationer. För att på ett realistiskt sätt kunna testa och utvärdera de algoritmer som utvecklats för att behandla detektion och klassificering av sådana förband finns därför ett behov av att de beskrivs av simuleringsmodeller. Dessa modeller ska beskriva hur förband och militära enheter uppträder i verkligheten, och också vara lättanvända och kunna återanvändas i nya simuleringar.

1.2 Modellering och simulering

Modellering och simulering innebär att skapa förenklade modeller av verkligheten eller en tänkt verklighet och sedan testa resultatet av ett händelseförlopp med dessa modeller. Modellering och simulering är en viktig arbetsmetod inom många områden, inte minst när det gäller utveckling av nya produkter och ny metodik. I stället för att bygga verkliga prototyper kan man testa sina teorier i en simulering. Om simuleringen är tillräckligt verklighetsnära så kan man bekräfta sina teorier eller upptäcka problem till en mycket lägre kostnad än om man hade byggt en verklig prototyp. Det kan också vara så att det inte är möjligt att testa sina teorier i verkligheten. I sådana fall är modellering och simulering en absolut nödvändighet för att besvara de frågor man har om vilka konsekvenser ett visst händelseförlopp har på den modell man byggt upp. Modellering och simulering används också som ett inlärningsverktyg för verksamheter som kan vara för dyra eller farliga att genomföra på riktigt, t.ex. flygningar vid pilotutbildning eller strategiska militärmanövrer.
Det är dock viktigt att validera sina resultat i så stor utsträckning som möjligt. En simulering som inte är tillräckligt lik verkligheten för att man ska upptäcka relevanta problem och svårigheter kan vara allt från värdelös till direkt farlig.

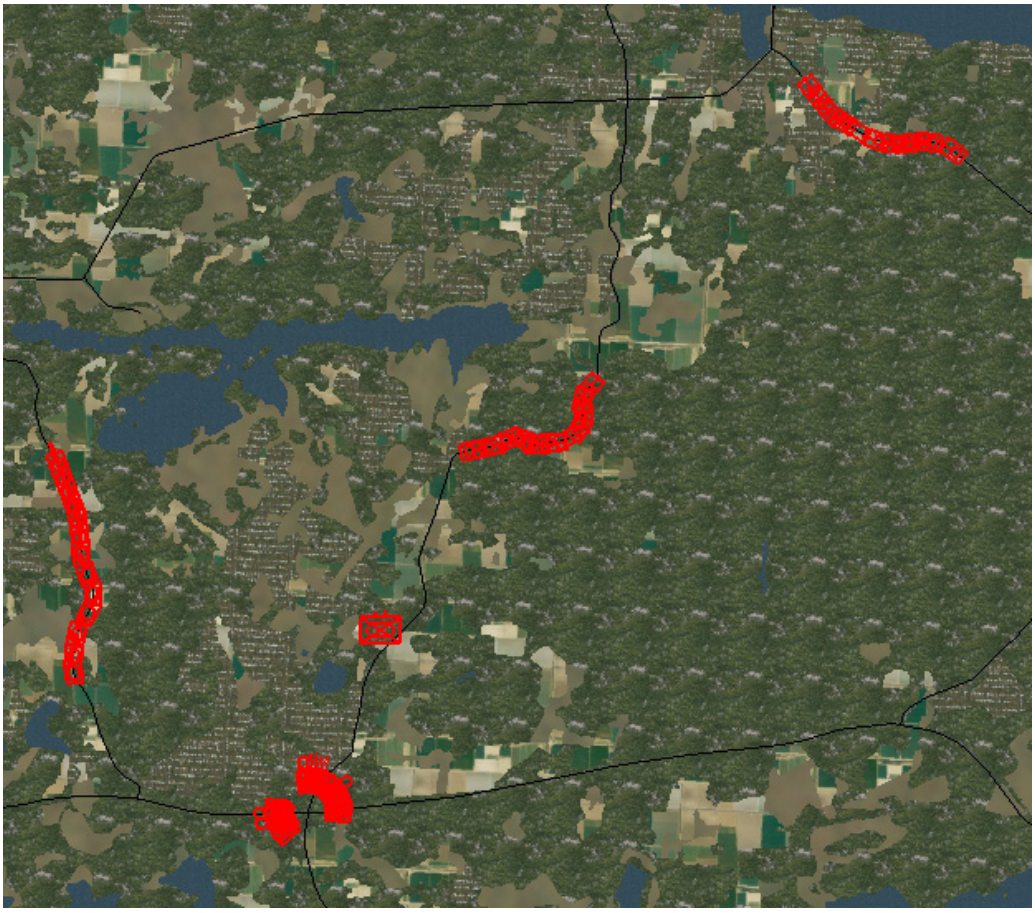
1.3 Syfte

Syftet med modellen är att skapa ett realistisk och lättanvänt sätt att modellera förflyttningar av fordonsbaserade truppförband enligt förutbestämda doktriner. Modellen är inte avsedd att ge en perfekt beskrivning av varje tänkbart beteende ett sådant förband kan ha. I stället är syftet att ge ett i huvudsak korrekt förflyttningsmönster som sedan kan användas för att testa detektionsalgoritmer och sensormodeller. Detta innebär att modellen ska kunna röra sig på ett

intelligent sätt och basera sina beslut på vad den vet om omgivningen, exempelvis vid vägval. Modellen ska också styras av ett begränsat antal parametrar och själv fylla i luckorna i det som användaren anger som indata.
Då modellen främst bör ses som en prototyp ska den också vara utbyggbar och tillräckligt generell för att kunna användas med ett flertal olika doktriner. Huvudmålet är att modellen ska kunna användas i en demonstrator som planeras vara klar till hösten 2003. Där ska den representera ett fientligt förband vars fordon detekteras av ett nätverk av sensorer. Den information sensorerna samlar in ska sedan behandlas av ett antal modeller/algoritmer för informationsfusion utvecklade inom det projekt modellen utvecklats för.

1.4 Scenario

Det scenario som är tänkt att användas i demonstratorn 2003 är en landstigning av en fientlig mekaniserad bataljon vid Kapellskär på Rådmansö. Den anfallande styrkan är en förstärkt förbataljon som är tänkt att innehålla 3 kompanier (11 fordon i varje uppdelade på 3 plutoner och en ledningsgrupp) med pansarskyttefordon och ett kompani med granatkastarvagnar. Styrkan är förstärkt med ett stridsvagnskompani och en luftvärnspluton med 4 robotvagnar. Det rör sig totalt om ca. 60 fordon. Den exakta sammansättningen kan dock variera i verkligheten. Den del av scenariot som är intressant för demonstratorn är de inledande händelserna efter landstigningen. Den angripande bataljonens uppgift i denna fas är att upprätta ett brohuvud och röra sig vidare mot Norrtälje. Bataljonen kommer att röra sig österut längs E18 och andra vägar. Vid ett givet klockslag kommer bron över Åkeröfjärden att sprängas och de delar av den angripande bataljonen som inte passerat kommer att tvingas gå runt Åkeröfjärden norrut. Då bara ett fåtal hemvärnssoldater finns i området kommer försvaret i denna inledande fas främst att bestå av spaning och klassificering av det angripande förbandet för att på lämpligt sätt kunna bekämpa detta.

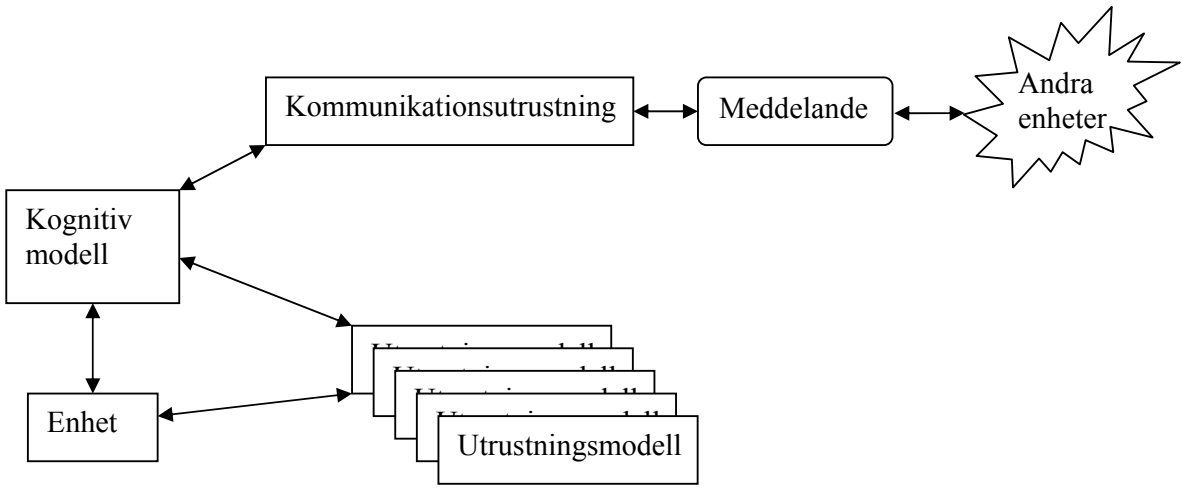


Figur 1: Framryckande styrkor på Rådmansö.

2 Problemformulering och definitioner

2.1 Ramverket FLAMES

Ramverket FLAMES som används i detta examensarbete är ett kommersiellt datorbaserat verktyg avsett att erbjuda en simuleringsmiljö snarare än att vara en simulator i sig [7]. Det är främst inriktat mot militära tillämpningar och används i ett 30-tal länder, både av militära och civila organisationer och företag. FLAMES är tänkt att användas för att simulera och visualisera stridsscenarier på en översiktlig nivå. Enheter som ingår i ett scenario kan kommunicera med varandra via radio och fientliga enheter kan bekämpas med vapenverkan. Man kan använda ett antal färdiga modeller i FLAMES och vill man ha enheter som har ett annat beteende kan man programmera egna moduler som länkas in i en egen version av FLAMES. Från början ingick enbart flyg och fasta markmål, men i senare versioner har det även kommit stöd för terräng och markfordon. Själva ramverket erbjuder förutom en databas för lagring av egna modeller ett grafiskt gränssnitt för scenariokonstruktion och uppspelning av scenarier. Det har stöd för terränghantering med TIN-modeller (Triangulated Irregular Network), meddelandehantering och funktionshantering inom och mellan de modeller som ingår i ett scenario. Rent praktiskt fungerar ramverket så att användaren själv skapar program för varje typ av enhet denne vill ha med. Dessa program är logiskt uppdelade i kognitiva modeller, utrustningsmodeller och meddelandemodeller.
Kognitiva modeller kan ses som modellernas ”hjärna” och beskriver en enhets beteenden och reaktioner på olika situationer. Utrustningsmodeller beskriver funktionaliteten hos fysisk utrustning som en modell har. Det kan vara köregenskaper för fordon eller förmåga hos en sensor att upptäcka andra enheter och så vidare. Meddelandemodeller används för att definiera en specifik typ eller klass av meddelanden som används när olika enheter vill kommunicera med varandra. Ett meddelande kan sedan förmedlas via ramverket i en så kallad Message Generation Point (MGP), som via avsändarens kommunikationsutrustning skickar meddelandet till mottagaren.



Figur 2: Samband mellan modelltyper i FLAMES

2.1.1 Skript

Man kan styra modellerna med hjälp av ett skriptspråk som skrivs in i ”Unit script” eller ”Dictionary script”. Det senare kan inkluderas i andra skript, vilket är praktiskt när flera enheter ska ha samma kommandon. Varje enhet måste ha ett eget ”Unit script”. Från dessa skript kan funktioner i kognitiva modeller eller i FLAMES kärna startas. Skripten kan också innehålla enkla villkor baserade på tid och händelser för att fördröja exekveringen.
Några exempel på kommandon i FLAMES skript:

```
INCLUDE inkluderar ett annat skript
EMPLOY PLATFORM tilldelar en plattform
ASSIGN COMMANDER tilldelar en ledare
LOCATE AT placerar enheten på en viss koordinat från början
START startar enheten vid en angiven tidpunkt
INITIATE startar en funktion i modellen
TRANSMIT meddelande som ska skickas
RECEIVE meddelande som ska tas emot
PERFORM funktion som ska utföras
```

Alla modeller lagras i FLAMES databas där de kan användas i de olika delprogram som ingår i FLAMES. För att skapa och använda ett scenario i FLAMES används i huvudsak fyra olika program.

2.1.2 FORGE

FORGE används för att skapa och redigera scenarier. Alla modeller som används lagras i olika dataset i en databas så att de kan återanvändas till nya scenarier. Det går förutom enheter och deras utrustning också att lägga till miljömodeller som t.ex. kartor, terräng, atmosfär, jordmodell etc.

2.1.3 FIRE

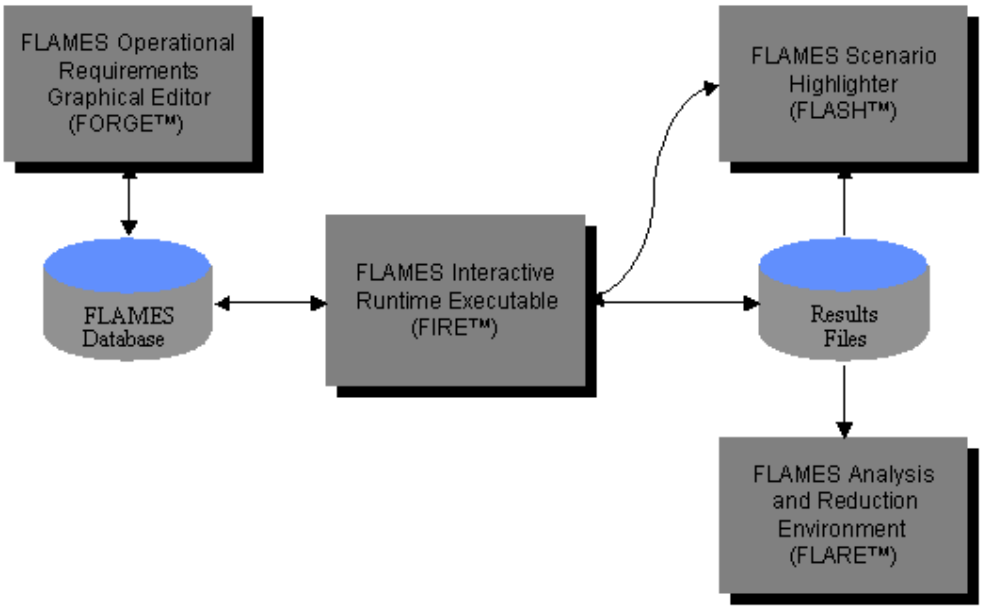
I FIRE exekveras scenarier som skapats med hjälp av FORGE och resultatdata sparas enligt användarens instruktioner.

2.1.4 FLASH

FLASH används till att spela upp scenarier antingen medan FIRE exekveras eller i efterhand.

2.1.5 FLARE

För att djupare analysera data från en scenariouppspelning används programmet FLARE.



Figur 3: Samband mellan delprogrammen i FLAMES.

2.1.6 Terrängstöd (FACT)

I den senaste versionen av FLAMES finns en utökning av stödet för terräng kallad FACT. Denna modul stödjer inläsning av TIN-modeller skapade med programvara från TerraVista, se nedan. I den version av FACT som existerar för närvarande saknas dock ett direkt gränssnitt mot dessa trianglar, vilket försvårar användandet av terrängdata. När arbetet utfördes gick
det endast att ange punkter och få ut data om dessa.

2.2 TIN (Triangulated Irregular Network)

En TIN-modell är en modell uppbyggd av trianglar där likheter mellan närliggande mätpunkter utnyttjas för att minimera antalet trianglar som behövs för att beskriva ett landskap. Detta innebär att stora trianglar används för att beskriva områden med små variationer i terrängtyp och topografi, medan mer varierade områden har flera mindre trianglar.

2.3 Doktriner

Modellen förutsätter att doktriner eller reglementen används, d.v.s. regelsamlingar för hur ett förband uppträder i olika situationer. I modellen anses alla förband eftersträva att följa en angiven doktrin. Hur exakt det går att beskriva ett verkligt förband med ett bestämt uppträdande i varje situation kan naturligtvis diskuteras. Det förutsätts dock i modellen att varje enhet, med de fysiska begränsningar den givits i form av t.ex. begränsad rörlighet och begränsad kunskap om omgivningen, gör sitt bästa för att följa den doktrin som används.

2.4 Marsch i formationer

Den huvudsakliga funktionaliteten hos modellen är att kunna genomföra marscher på ett trovärdigt sätt. För att göra detta krävs förståelse för hur en fordonsmarsch genomförs i verkligheten och eventuella variationer i utförandet av den. De faktorer som påverkar

marschen är främst vem som genomför den, situationen som den genomförs i och hur eventuella problem som uppstår under marschen löses.
Metodiken som används för att förbereda och genomföra en marsch bygger främst på material hämtat ur [3], [4] och [5] och har kontrollerats med personer med militär bakgrund och som har erfarenhet som marschledare.

En marsch startar normalt från en grupperingsplats eller annan samlingsplats och passerar genom en kolonnbildningspunkt, där själva kolonnen bildas. Denna punkt kan t.ex. vara en närliggande korsning eller påfart till en väg.
Marschen organiseras av en marschledare, som normalt är en person ur förbandets ledning. Denna person ställer upp ordningen för marschen och ser till att alla är på rätt plats.
Den huvudsakliga framryckningsmetoden under en marsch är att färdas på kolonn längs en väg, eventuellt med flankskydd i form av stridsvagnar eller pskvagnar (pansarskyttevagnar) som färdas vid sidan av vägen. Ett förband av bataljons storlek kommer dessutom att ha en stridspatrull som färdas några kilometer framför huvudkolonnen, och i fallet med en mekaniserad bataljon består den av en pluton pskvagnar. Det finns normalt även ett förkompani som ligger ca 3 km framför huvudkolonnen och som normalt är ett mekaniserat skyttekompani. Efter förkompaniet kommer huvudkolonnen med bataljonsledningen och resterande förband, utom en pluton pskvagnar som används som köpluton och färdas ca 3km bakom huvudkolonnen.

Figur 4: Marsch i kolonn.

Om kolonnen stöter på fientliga förband ska dessa om möjligt kringgås. Samma sak gäller om vägen är blockerad eller av annan anledning är oframkomlig. Förband marscherar normalt under radiotystnad och radiotystnaden bryts bara om förbandet upptäcks eller angrips av en fiende, eller om en katastrof av något slag inträffar, t.ex. att ett fordon skadas och besättningen måste undsättas. Så länge det inte föreligger något hot mot kolonnen kommer den att färdas i sin ursprungliga formation och under radiotystnad.
När man nått målet för marschen så ställer förbandet upp sig i en gruppering eller stridsställning beroende på hotläget vid destinationen.
För att samordna flera marscher används speciella transportförband, som bl.a. ansvarar för att fördela trafiken över tillgängliga vägar, leda om trafik runt hinder och ta hand om eftersläntare.

2.5 Samverkan mellan förband

Förbandet som modelleras är strikt hierarkiskt och det kommer därför inte att behövas någon direkt samverkan mellan förband på samma hierarkiska nivå. Om en befälhavare inte själv kan fatta ett beslut, kommer denne att skicka problemet till nästa nivå i hierarkin tills en chef med tillräckliga befogenheter för att lösa problemet kan ta ett beslut. Vid eventuell samverkan med transportförband kommer den lokale transportchefen att vara överställd den marscherande enhetens chef och samma metodik kan tillämpas även här.

2.6 Hotsituation

Den situation som det modellerade förbandet befinner sig i innebär att det finns ett mycket starkt hot från framförallt angrepp med flyg och artilleri, men också från fientliga markförband. Detta innebär att förbandet måste kunna detektera och bedöma hot i dess omedelbara närhet, men då strid inte kommer att ingå i simuleringen så kommer det inte att behövas någon modellering för hur förbandet ska svara på eventuella hot.

2.7 Markfordon

Flera olika slags fordon ingår i förbandet, t.ex. stridsvagnar, stridsfordon och lastbilar. De har olika förmåga att ta sig fram i terräng bl.a. beroende på om de är hjul- eller bandgående. Dessutom har de olika accelerationsförmåga, maxhastighet och svängradie. Även om inte olika fordonstyper särbehandlas vid beräkning av rutter i prototypen bör det finnas stöd för att göra detta senare.

3 Kravbeskrivning

3.1 Intelligent uppförande

Enheternas uppförande ska vara trovärdigt, både när det gäller förband och enskilda fordon.

3.2 Rörelse med hänsyn till doktriner

Enheternas beteende ska regleras av doktriner.

3.3 Signalering

De olika enheterna ska kommunicera med hjälp av det förutbestämda gränssnitt som finns i FLAMES, via radio eller muntligen.

3.4 Vägval i terräng och på vägar

För att enheterna ska kunna förflytta sig realistiskt och modellen samtidigt ska vara lättanvänd, måste den självständigt kunna räkna ut hur förflyttning ska ske till en specifik destination. Förband ska självständigt kunna välja vilken väg de ska följa för att snabbast ta sig fram till målet. Vägvalet ska baseras på hur vägnätet i den omgivande regionen ser ut, och eventuellt på var fientliga förband siktats. Om och när det är möjligt att utläsa terrängdata med ett gränssnitt som ger mer exakt terränginformation ska vägval och förflyttning i terräng läggas till.

3.5 Undvikande av hinder

Hinder som upptäcks längs vägen ska både kunna detekteras och kringgås. Detta gäller även hinder som uppstår dynamiskt under simuleringen. I demonstrationsscenariot rör det sig främst om den bro som sprängs när ett anfallande kompani bara delvis passerat.

3.6 Utbyggbarhet

Olika hotsituationer bör tänkas igenom så att det vid behov är möjligt att bygga ut modellen för att klara sådana situationer. Man kan också vilja ha en uppdelning av förband på olika marschvägar för att undvika trafikstockningar och onödiga truppkoncentrationer, ett problem som i vanliga fall skulle ha lösts av en transportchef. Även om transportförband inte implementeras i prototypen ska det vara möjligt att lägga in detta beteende i modellen senare.

4 Avgränsningar

4.1 Terräng

På grund av den begränsade åtkomsten av terränginformation har vi valt att inte implementera några terrängalgoritmer. Prototypen kan därför bara använda sig av vägnätet för att räkna ut sina marschrutter. Det begränsar också möjligheten att modellera realistiska grupperingar, att ta skydd vid sidan av vägen vid flyganfall och möjligheten att rycka fram på linje.

4.2 Strid

Hantering av stridssituationer ingår inte i förflyttningsmodellen, och inga hänsyn tas till fientliga förband i detta avseende.

4.3 Kommunikation

I modellen förutsätts att alla enheter alltid kan kommunicera via radio utan att påverkas av vare sig störningar eller avstånd.

4.4 Fordonsmodellen

Köregenskaper som sladd eller svårighet att följa vägen vid för skarpa kurvor ingår inte i fordonsmodellen. Från början begränsades den största möjliga vridningen fordonet kunde göra per tidsenhet av dess svängradie, hastighet och acceleration. Eftersom det är viktigare att fordonen håller sig på vägarna och åker i rätt riktning vid normal körning än att kurvtagningen är helt realistisk har vi valt att låta fordonen vrida sig momentant i svängarna. Fordonsmodellen tar inte hänsyn till markegenskaper som terrängtyp eller lutning när möjlig hastighet beräknas. Eftersom dessa egenskaper inte används när fordonets rutt räknas ut skulle detta medföra att fordonen fastnar vid hinder som inte ingår i den kognitiva modellens uppfattning av världen.

4.5 Vägval

Enheter förutsätts ta den kortaste vägen till destinationen om inget annat specifikt angetts. De fall där en enhet inte ska ta den närmaste vägen måste specificeras av användaren.

4.6 Realtid

Det finns inga krav på att modellen ska fungera i realtid, då de övriga modellerna som den ska testas mot typiskt exekverar avsevärt långsammare än realtid. Syftet med simuleringarna innebär inte heller några sådana krav.

4.7 Utbyggbarhet

Då modellen är tänkt som en prototyp till en mer avancerad version ska den vara fullt utbyggbar inom de begränsningar som ges av FLAMES. Detta innebär att den i så stor utsträckning som möjligt ska vara modulbaserad och använda sig av de i FLAMES specificerade gränssnitten för kommunikation mellan olika modeller. Modellen kommer dock

inte att ge något som helst stöd för användning i andra simuleringsramverk än FLAMES eller för fristående användning.

5 Metodbeskrivning

5.1 Förstudie

I den inledande litteraturstudien var det framförallt information om ramverket FLAMES i sig och information om hur en marsch genomförs och vilka doktriner som används som vi fördjupade oss i.

5.1.1 Inlärning av FLAMES

För att kunna genomföra examensarbetet krävdes det först en ganska djupgående förståelse för hur FLAMES används och är uppbyggt. Då det fanns en ganska noggrann kurs i användandet av FLAMES gick denna del relativt snabbt. Att komma på djupet med hur programmet är uppbyggt och hur de nya moduler som skulle skrivas passade in var däremot inte lika väldokumenterat och tog längre tid att sätta sig in i.

5.1.2 Genomförande av marsch

För att få en så komplett bild som möjligt av hur marscher genomförs i verkligheten studerades främst de militära reglementen från svenska armen som vi ansåg relevanta och annat publikt material om utländska förbands uppträdande och tillvägagångssätt som t.ex. [4] och ett antal rapporter. De resultat som sedan kunde extraheras ur dessa skrifter kontrollerades med personer med militär bakgrund på FOI för att få en uppfattning om hur nära verkligheten de låg. Resultatet blev en konceptuell modell för hur en marsch genomförs, inkluderad i bilaga A.

5.2 Specificering av modeller

Den ursprungliga specificeringen av modeller innebar att det skulle konstrueras två egna modeller, en generell kognitiv modell och en generell fordonsmodell som skulle användas av alla fordon. I övrigt var avsikten att använda de färdiga FLAMES-modeller som fanns från början. Senare visade det sig att det fanns ett behov av en egen sensor och en mer lättanvänd modell för hantering av radio, varvid dessa implementerades.

5.3 Designprocess

5.3.1 Utvecklingsmetodik

Den utvecklingsmetod som användes för modellen var en evolutionär process med gradvis mer kompetenta prototyper. Då det i början av arbetet inte var klart hur mycket av de utlovade uppdateringarna av FLAMES som skulle finnas tillgängligt blev denna metodik en nödvändighet. De första prototyperna var väldigt inriktade mot gruppbeteende då det inte fanns tillgång till vare sig vägar eller terrängdata. När terräng- och väginformation tillkom gick tonvikten över på vägval och förflyttningar.

5.3.2 Utvecklingsmiljö

Alla implementationer har gjorts i C++ och Microsoft Visual Studio har använts som utvecklingsmiljö.

6 Implementation

6.1 Programstruktur

Den struktur programmet har fått har till stor del dikterats av hur FLAMES är uppbyggt. Vi har försökt göra modellen så objektorienterad som möjligt för att andra lättare ska kunna bygga vidare på den eller byta ut delar av den. Modellen består av en kognitiv modell "FOIIF Ground Vehicle Commander", en utrustningsmodell "FOIIF Ground Vehicle", en sensormodell ”FOIIF GVC Eyes” och en radiomodell ”FOIIF Simple Radio Manager”. Se bilagorna B och C för en mer detaljerad beskrivning av modellerna än nedanstående.

6.1.1 FOIIF Ground Vehicle Commander (GVC) - Kognitiv modell

Den kognitiva modellen motsvarar enhetens hjärna och fattar alla dess beslut. Den primära funktionalitet som ingår i denna modell är att bygga upp en befälshierarki och utföra uppdrag enligt instruktioner. Uppdrag i modellen innefattar att planera och genomföra marscher till givna destinationer. En marsch förbereds på så sätt att en marschledare räknar ut rutter och förmedlar dessa till underlydande enheter. Dessa förmedlar sedan order till sina underordnade o.s.v. Den kognitiva modellen består av flera processmetoder, som anropas antingen via FLAMES skriptspråk eller av andra processmetoder. En utförlig beskrivning av dessa processmetoder och deras funktion finns i bilaga B.

6.1.2 FOIIF Ground Vehicle (GV) - Utrustningsmodell

Utrustningsmodellen beskriver en enhets fysiska interaktion med omvärlden. Den kognitiva modellen meddelar utrustningsmodellen en rutt som ska följas när en marsch förbereds. Under marschens gång anges sedan önskade hastighetsändringar för att antingen följa efter ett fordon på lagom avstånd eller för att anpassa sig till annan trafik i korsningar. Utrustningsmodellen sänker själv hastigheten inför kurvor och vid marschens destination. När fordonet är framme skickas ett meddelande till den kognitiva modellen som talar om att målet är nått.

6.1.3 FOIIF GVCEyes - Sensormodell

Sensormodellen kan upptäcka hinder, fiender och utslagna enheter inom en angiven radie från den egna positionen.

6.1.4 FOIIFSRM – Radiomodell

Radiomodellen låter användaren tilldela nätverk och kanaler för kommunikation mellan enheter.

6.2 Funktionalitet

Detta avsnitt beskriver de huvudsakliga funktioner som modellen har och hur de fungerar.

6.2.1 Initiering

GVC-modellen initieras med funktionen INIT, som bygger upp en nodkarta och en befälsstruktur för förbandet. I denna funktion tilldelas också namn på förband och rang för enskilda enheter. Namnen på förband används för att högre chefer ska kunna identifiera underlydande enheter när de förmedlar order till dessa. Rang används när förlorade enheter

ska ersättas. Även namn på den minsta logiska enhet ett fordon tillhör kan tilldelas här. Namnets funktion i modellen är endast att erbjuda en möjlighet att lättare identifiera enheter vid utskrifter. I framtida utbyggnader är den tänkt att användas för att sammanbinda och identifiera t.ex. ledningsgrupper för ett förband.

6.2.2 Ordergång

Modellen är strikt hierarkisk och alla order passerar varje nivå i befälshierarkin. Det enda undantaget är om hinder upptäcks under pågående marsch. Då förmedlas instruktioner om ny färdväg via fordon i närheten eftersom det normalt råder radiotystnad under marsch och befälhavaren kan vara svår att nå på annat sätt.

6.2.3 Uppdrag

Modellen använder ett system med uppdrag för att lösa sina uppgifter. Då modellen inte behandlar stridsituationer utan bara förflyttningar är de enda uppdrag som implementerats marscher av varierande karaktär.
För att hålla reda på ordningen på uppdragen används ett system med olika tillstånd. Dessa tillstånd kan utnyttjas för att få en godtycklig ordingsföljd av uppdrag, där senare uppdrag inte utförs innan de föregående är klara. Flera tidigare uppdrag kan också tillsammans sätta ett tillstånd när alla är klara. Uppdrag kan också utföras frånkopplat från högre befäl, i vilket fall den frånkopplade enheten/förbandet inte deltar i de uppdrag som de större förband den ingår i utför. Om varken tillståndssystemet eller frånkoppling används är det dock alltid det senast beordrade uppdraget som gäller.

6.2.4 Marscher

En marsch genomförs i modellen genom att funktionen SET_MARCH anropas i ett enhetsskript, antingen för den högsta chefen i den marscherande enheten eller för en överordnad chef. När startvillkoren i form av eventuella tillstånd uppfyllts inleder marschledaren, som är den högste befälhavaren i den marscherande enheten planeringen av marschen. Detta är en mindre förenkling av verkligheten eftersom det ofta är en ställföreträdande chef som leder marschen. Då detta inte har någon praktisk betydelse i modellen kan denna förenkling anses vara motiverad. Den planering som genomförs är i första hand att marschledaren räknar ut den rutt marschen ska följa samt bygger upp den formation som marschen ska använda. De underlydande enheterna meddelas sedan hur marschen ska genomföras. De fordon som är långt från kolonnbildningspunkten där marschen ska starta tar sig först dit. När de är tillräckligt nära meddelar de att de är redo att marschera. Om starttid angetts inleds marschen när denna tid har nåtts, annars omedelbart när alla samlats. Se även bilaga B för detaljerad information om vilka parametrar som används för att beskriva en marsch. Det går även att ange speciella kriterier för vilken väg en viss marsch ska följa genom att använda en ruttfil, se vägföljning nedan.

6.2.5 Lägesbedömning

Enheterna håller löpande koll på sin omgivning genom att köra en kontinuerlig lägesbedömningsprocess. Denna process kontrollerar om det finns något i omgivningen som enheten måste hantera. Modellen hanterar förluster och hinder längs vägen i denna process. Den detekterar också fiender, men då strid inte ingår i modellen åtgärdar den inte dessa. Enheterna har sensorer som upptäcker allt inom ett visst avstånd, och det är främst informationen från dessa som behandlas.

6.2.6 Ersättning av förlorade befäl

Ett befäl som slås ut kan ersättas genom att någon underlydande befordras. Det görs inte omedelbart, utan först när någon i förbandet upptäcker att dess befäl är försvunnet eller utslaget. Att en enhet saknas kan upptäckas genom att den inte svarar på radioanrop, eller att någon hittar vraket efter dess fordon. Ersättning sker genom att den förlorade enhetens närmaste chef meddelas och utser en efterträdare, som får ett befordringsmeddelande. Detta innebär att modellen kan ersätta alla befäl utom den högsta chefen så länge det finns underlydande enheter kvar.

6.2.7 Undvikande av hinder

Modellen upptäcker hinder med hjälp av en sensor och räknar ut hur den ska ta sig runt dem. Om en enhet upptäcker ett hinder som den inte meddelats om av någon annan enhet så räknar den ut en ny rutt från sin nuvarande position till målet. Dessutom meddelar enheten den nya rutten till ett eventuellt efterföljande fordon, som i sin tur vidarebefordrar meddelandet bakåt i kolonnen. Fordonen i kolonnen kommer att behålla sin inbördes ordning genom att alla är tvungna att köra fram till den punkt där rutten räknades om. Det blockerade vägavsnittet som detekterats klipps också bort ur nodkartan för att inte tas med i framtida vägvalsberäkningar. Återställningsinformation av nodkartan lagras ifall hindret skulle försvinna. För att undersöka om ett hinder ligger i ett fordons färdväg testas dess position mot koordinaterna i den rutt fordonet följer. Om dessa överlappar varandra så byter fordonet färdväg.

6.2.8 Kollisionstest

Fordonen undviker att kollidera i korsningar och vid möten i terrängen genom att de testar om det finns andra fordon som står på eller är på väg in på den rutt som fordonet för tillfället följer. Om det finns risk för kollision tillämpas högerregeln i första hand. Om ett fordon som egentligen skulle lämna företräde fysiskt står i vägen så åker det dock vidare för att undvika låsningar. Kollisionstestet genomförs genom att skapa begränsningsboxar runt fordonen och förlänga dessa längs rutten för fordonet som genomför testet och i färdriktningen för fordon i närheten. Begränsningsboxarna förlängs proportionellt mot den hastighet fordonet har för att ge tillräcklig tid att bromsa.

6.2.9 Vägföljning

Som utgångspunkt för att göra en rutt finns den egna positionen, en destination, eventuellt ett vägnät samt eventuellt förutbestämda vägval givna. Eftersom det inte finns tillgång till något bra gränssnitt mot terränginformationen har arbetet koncentrerats på att utnyttja vägnätet. Vid programmets start byggs en nodkarta upp utgående från den information om vägnätet som lästs in i FLAMES, där vägarna består av ett antal koordinater (vägpunkter). När en rutt ska räknas ut söks de punkter upp på vägnätet som ligger närmast start- och slutpositionen upp och lagras temporärt i form av noder i nodkartan. Det behöver inte vara på en vägpunkt, utan kan vara en godtycklig punkt på vägsträckan. Därefter används Dijkstras algoritm[8] för att räkna ut kortaste vägen mellan start- och slutnoden. Om dessa noder inte är förbundna med varandra kommer algoritmen att misslyckas, vilket resulterar i att fordonet kör raka vägen över terrängen från start- till slutpunkten. Om användaren inte vill att rutten ska vara den kortaste vägen till destinationen kan vägvalsinformation anges i en textfil, där tre kommandon kan användas: **COORDINATE, INTERSECT** och **SHORTEST PATH**. **COORDINATE** ska följas av ett latitud- och ett longitudvärde, som anger nästa punkt i rutten. **INTERSECT** ska följas av ett vägnamn på en väg som måste finnas med som en delsträcka i rutten. **SHORTEST PATH** ska följas av ett latitud- och ett longitudvärde som måste finnas med i rutten. **INTERSECT** och **SHORTEST PATH** kan även följas av **ON** och ett vägnamn som

anger nästa väg. **INTERSECT** kan också följas av **NEAR** och en koordinat för att ange en speciell korsning om det finns flera att välja på, annars kommer den närmaste att väljas. De kommandon som beskrivits anges enligt följande syntax i textfilen:

COORDINATE lat lon
INTERSECT vägnamn [**ON** vägnamn] [**NEAR** lat lon]
SHORTEST PATH lat lon [**ON** vägnamn]

7 Resultat

7.1 GVC-modellen

GVC-modellen har i stort sett de egenskaper som specificerades i beskrivningen av examensarbetet.

- Marschera i formation.
- Göra vägval, helt fritt eller efter instruktioner i textfil.
- Undvika hinder genom att byta väg.
- Detektera fiender.
- Ersätta förlorade befäl.
- Undvika kollisioner.

Den största inskränkningen i funktionalitet är att stöd för förflyttning i terräng helt saknas, se diskussion nedan.
Modellen använder sig inte heller av transportförband för att fördela eller leda om trafik, utan detta lämnas till användaren eller som en möjlig utbyggnad.

7.2 Diskussion

Som alla ramverk innebär FLAMES en avvägning mellan att förenkla för användaren och erbjuda största möjliga flexibilitet. Den kanske största fördelen med att använda ett ramverk av den här typen är att det tvingar utvecklare att skapa modeller som kan fungerar modulvis.
Den kanske största begränsningen i FLAMES när det gäller den prototyp som framställts är att det saknats ett meningsfullt gränssnitt mot den nya terrängrepresentation som modellen är tänkt att använda. Terrängen är representerad av en TIN-modell, men det har inte gått att få tillgång till den kompletta triangelinformationen från FLAMES, utan det var bara möjligt att fråga efter marktyp och höjddata i enskilda punkter.
Det skriptspråk som FLAMES använder för att beskriva scenarier har också vissa brister. De begränsningar som berör oss är främst att det inte på ett adekvat sätt går att göra funktionsanrop blockerande så att exekveringen av skriptet stoppas tills funktionen är avslutad.
Att det endast är möjligt att mata in flyttal och strängar som parametrar till de funktioner som anropas innebär också vissa problem, inmatningen blir t.ex. onödigt omständlig. Dessa problem är inte oöverstigliga, men de innebär att mycket tid fick läggas på att lösa dem, tid som annars kunde ha använts till att förbättra modellen.
Alla funktioner i FLAMES som var nödvändiga för det här arbetet fanns inte tillgängliga från början, utan det har kommit flera uppdateringar efter hand. Detta har gjort det svårt att veta vad som skulle bli möjligt att implementera och arbetet har efter hand koncentrerats till de områden som bara krävt funktionalitet som varit tillgänglig. FLAMES minneshantering har skapat en del problem med svårupptäckta minnesläckor eftersom alla funktioner inte varit konsekventa. En del funktioner destruerar minnet de använt, medan andra inte gör det trots att skillnaden inte framgår av dokumentationen. Ett annat problem med att arbeta mot ett system under utveckling har varit att dokumentation ibland har saknats eller varit inaktuell.

7.3 Testning

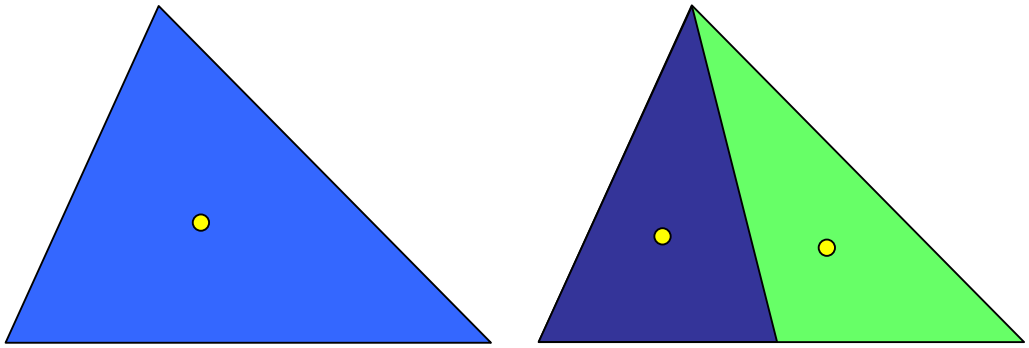
Under utvecklingsarbetet har testning skett löpande genom körningar av scenarier med varierande storlek och innehåll. Hur stora fordonsgrupperingar som modellen klarar av gick tyvärr inte att testa då ett internt problem i den pre-release av FLAMES 5.1 som användes satte en maxgräns på ungefär 500 fordon. Minnesåtgången för 500 fordon hamnar runt 1,4GB, vilket gör att modellen blir praktiskt oanvändbar för fler fordon. Detta medför dock inte några problem med att simulera det tänkta scenariot då det endast innehåller en förstärkt bataljon med ca. 60 fordon och några enstaka enheter som ska detektera bataljonen.
Beräkningsmässigt är den tyngsta delen av modellen fordonens sensorer och deras detektion av närliggande objekt. Vid 500 fordon som alla kan se varandra står sensorerna för ca 90 % av den totala beräkningstiden och är kvadratisk i komplexitet i förhållande till antalet fordon. Det som i första hand påverkar exekveringstiden är sensorernas räckvidd och hur utspridda fordonen är medan minnesåtgången främst styrs av sensorernas lagring av observationer, antalet marscher som genomförs samtidigt och hur befälshierarkin ser ut. Modellens beräkningskostnad vid denna scenariostorlek är dock försumbar i jämförelse med de algoritmer den ska testas mot. Inte heller minnesåtgången för vår modell utgör ett problem vid denna storlek. Det skulle vara möjligt att optimera prestandan för programmet avsevärt genom att flytta sensorberäkningarna till den kognitiva modellen, men detta skulle sabotera objektorienteringen i modellen.

Figur 5: Hoppande T90.

8 Möjliga förbättringar/vidareutvecklingar

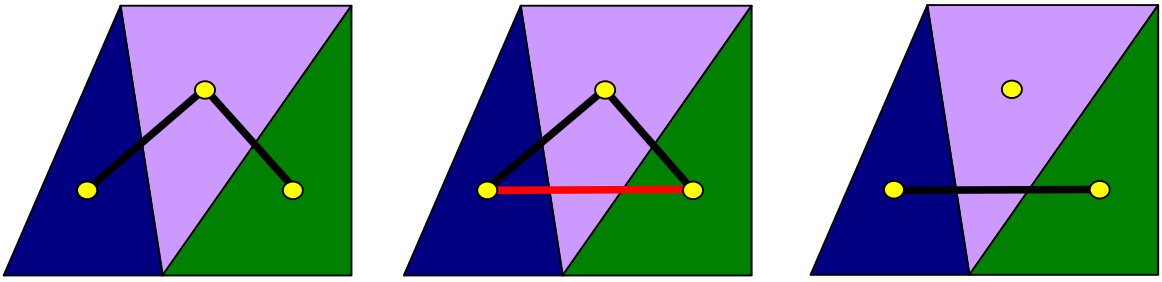
8.1 Vägval med hänsyn till terräng

En idé vi hade till vägvalsalgoritm var att behandla terrängen tillsammans med vägnätet. En nodkarta skulle skapas utgående från triangelinformationen. En nod sätts i centrum av varje triangel. Stora trianglar delas i två delar mitt på den längsta sidan. Om triangeln fortfarande är för stor görs detta rekursivt tills alla trianglar är tillräckligt små. Hur stor en triangel får vara avgörs med avseende på vad man anser sig ha råd med i fråga om antal noder och kanter som genereras.



Figur 6: Uppdelning av stora trianglar.

Varje nod förbinds sedan med de noder vars trianglar gränsar till den triangel där noden är placerad. Kostnaden för varje kant baseras på lutning och terrängtyp. Hjälpnoder sätts in där två grannnoder inte går att förbinda med en rak kant utan att passera över andra trianglar. För att beräkna en rutt utgående från nodkartan som skapats används A* [6]. A* är en algoritm som liknar Dijkstras algoritm, men den har fördelen att man inte behöver undersöka alla noder i nodkartan, utan kan tillåta en mer lokalt optimal lösning. Detta görs genom att noder som ligger längre bort än ett uppskattat längsta avstånd inte behöver användas i beräkningen. Vägarna skulle automatiskt kunna bli föredragna eftersom man kan ange att kostnaden för att färdas på terräng av vägtyp är lägre än all annan terrängtyp.
När nodkartan är klar räknar man ut enskilda rutter med hjälp av A*. Sedan optimeras dessa genom att man gallrar bort onödiga noder. Detta går till så att för varje nod med minst en nod före och en nod efter sig i rutten testas om noden kan plockas bort och ersättas med en rak kant mellan noderna före och efter. Ändringen genomförs bara om den nya kanten både är giltig, d.v.s. bara går över terrängtyper tillåtna för de fordon som ska marschera, och har en lägre kostnad än de tidigare kanterna. Detta gör att man också till stor del undviker de sicksackformade rutter som kan uppstå i en rutt baserad på en nodkarta. Gallringen av noder kan fortsätta tills det inte längre går att genomföra några förbättringar eller avbrytas när ett visst antal tester eller förbättringar genomförts, baserat på vilken beräkningskostnad man anser sig kunna klara av.



Figur 7: Optimering av rutt. Om en kortare tillåten rutt existerar mellan en nods närmaste grannar, optimeras noden bort.

Genom att använda ett genomtänkt stoppvillkor för A* och se till att största storleken på trianglar sätts beroende på kartstorleken och det totala antalet trianglar kan man ganska noggrant kontrollera att vägvalet blir så bra som möjligt utan att överbelasta systemet. Algoritmen kommer inte alltid att generera den optimala rutten, men med tillräckligt små nodavstånd och tillräckligt mycket gallring av noder kan man få ett relativt litet fel. Det är också möjligt att genom mer avancerad efterbehandling få mer exakta lösningar. Genom att i stället för att ta bort noder testa hur nära de kan flyttas för att bilda en rak linje mellan föregående och efterföljande nod kan man få en ännu bättre lösning, fast till en högre beräkningskostnad [2]. Eftersom den kortaste rutten väljs ut före optimeringen kan algoritmen dock missa rutter som efter optimering skulle varit kortare än den utvalda och optimerade rutten. Storleken på detta fel bestäms endast av hur tätt noderna är placerade runt hinder som rutten kringgår.

8.2 Automatisk förbandsgenerering

Något som skulle underlätta väldigt mycket för användaren vore om man kunde generera hela förband automatiskt. I nuläget måste man skriva ett särskilt skript för varje enhet där man tilldelar fordonsplattform, startposition, ett antal processmetoder, etc. I stället för att generera varje enhet separat skulle man kunna skapa ett skript bara för den högste chefen i varje styrka. I detta skript kan man sedan deklarera ett antal underlydande förband av olika typer som då genereras automatiskt. På detta sätt skulle man väldigt enkelt kunna skapa avancerade scenarier med ett stort antal inblandade fordon. Detta kräver att mallar skapas för ett antal olika förbandstyper, och att ett system för att skapa nya mallar implementeras.

8.3 Grupperingar

De grupperingar som finns för tillfället är varken realistiska när det gäller uppställning eller i att de undviker att ställa sig på olämplig terräng. När ett mer avancerat gränssnitt mot terrängen finns tillgängligt kan man i stället ställa upp dem längs existerande eller automatgenererade terrängslingor. De bepansrade fordonen kan då placeras i försvarsställningar på lämpliga ställen runt grupperingen. Det bör också gå att skapa tillfälliga grupperingar längs den väg fordonen färdas på.

8.4 Försvarsåtgärder

Modellen reagerar i nuläget inte på anfall eller upptäckta fiender eftersom det inte krävs i det projekt den först och främst ska användas i. För att den ska kunna återanvändas i andra projekt bör det läggas till försvarsåtgärder när fordon angrips, möjlighet att ta skydd vid flyganfall eller liknande situationer.

8.5 Samordning av förband

En möjlig utbyggnad för att få en realistisk fördelning av marschvägar och kunna leda om trafik på ett bra sätt är att införa någon form av trafikkontroll. Denna modell skulle motsvara en transportchef i verkligheten. En sådan modell skulle kunna dirigera trafiken för att automatiskt fördela marscher och transporter över vägnätet och undvika trafikstockningar. Den måste då interagera med vägvalsalgoritmen för att förlägga marscher längs specifika vägar. Detta kan göras genom att dela upp ruttberäkningarna i flera etapper, där de passerar vissa kontrollpunkter placerad på den tilldelade vägen, eller genom att förbjuda andra vägar.

9 Referenser

[1] Lena Jönsson Sammanställning av underlag för projektet ”Taktisk informationsfusion” 2001
[2] M. de Berg, M. van Kreveld, M.Overmars, O Schwarzkopf Computational Geometry, 2nd ed. 2000.
[3] Försvarsmakten Transportreglemente för Försvarsmakten Landtransporter 1998
[4] Försvarsmakten Utländska stridskrafter 1993
[5] Försvarsmakten BrigR A Bat 2000
[6] George F. Luger & William A. Stubblefield Artificial Intelligence 1998
[7] Ternion Corporation http://www.ternion.com/
[8] Gilles Brassard & Paul Bratley Fundamentals of Algorithmics

Bilaga A. Konceptmodell för fordonsmarsch

Detta dokument är tänkt att översiktligt beskriva de valmöjligheter och handlingar som ska beaktas i modellen för fordonsförflyttningar.
Förflyttning av förband i bataljons storlek eller mindre kommer i vår modell att begränsa sig till att marschera kortare sträckor inom ett befälsområde i en krigsliknande situation. Varje marsch kommer att ledas av den högsta chef som ingår i den enhet som skall förflyttas, hädanefter kallad marschledare. Marschen ska kunna omdirigeras och avbrytas dynamiskt och den ska också kunna anpassa sig efter ett ändrat hotläge.

Marschledaren kan antingen själv eller via högre befäl bestämma riktlinjer för hur marschen ska genomföras. Om fientliga förband finns i området ska marschen läggas så att dessa undviks. Om färdvägen blockeras ska hindret om möjligt kringgås. Detta sköts av de enheter som upptäcker hindret. I verkligheten antas att fordonen samlas vid hindret och att den högsta närvarande chefen tar beslut om kringgång. Då vem som tar beslutet utåt sett saknar betydelse kan detta förenklas till det första fordon som anländer till platsen.

Strid kommer inte att ingå i modellen, men ska vara en möjlig utbyggnad, vilket gör att de valmöjligheter som strid innebär ska beaktas.

Doktriner

Marschen organiseras med avseende på en doktrin, och antas följa denna relativt exakt.

Marschledares aktiviteter före och under marsch

- Bestäm/Motta marchberedskap.
- Bestäm/Motta mål.
- Bestäm underliggande förbands mål.
- Bestäm/Motta kolonnbildningspunkt (startpunkt).
- Bestäm/Motta formation/marschordning.
- Beräkna/Motta rutt(er) (med hänsyn till andra transporter).
- Bestäm/Motta hastighet.
- Bestäm/Motta sluttid.
- Beräkna tidsåtgång (ev. med hänsyn till annan trafik).
- Beräkna/Motta starttid.
- Beordra förband som ska ingå i marschen till startpunkten (Se marschorder nedan).
- Invänta starttid.
- Ge klartecken till enheter enligt marschordning.
- Hantera händelser längs vägen (se nedan).
- Återsamla förband vid mål.
- Klar!

Ordning för marsch för underordnad enhet

- Ta reda på vilka enheter som ingår i egen grupp.
- Utse ledare och ställföreträdare.

- Ta reda på målen (invänta order).
- Bestäm marsch- förflyttningsordning enligt order.
- Ta sig till startpunkt.
- Meddela befäl att man är marschberedd.
- Vänta på klartecken.
- Starta marschen.
- Hantera händelser längs vägen. (se nedan)
- Meddela befäl när målet nåtts.
- Klar!

Händelser/Störningar längs vägen

Enheter kan utföra en eller flera av de åtgärder som beskrivs vid varje punkt beroende på vad dess egen kognitiva modell beslutar och vad dess chef beordrar.

- Vägen blockerad
  - Byt väg är det alternativ som normalt bör föredras.
  - Rådfråga chef.
  - Reparera vägen (Detta innebär att förbandet avbryter marschen och övergår i stridsgruppering).
- Fiende upptäckt (Behandlas ej av modellen, men ska vara en möjlig utbyggnad).
  - Kringgå fiende, vilket normalt är att föredra.
  - Fortsätt.
  - Fråga chef.
  - Anfall, marschen går i så fall över i manövergruppering och vid anfall i stridsgruppering.
  - Ta skydd, att föredra om fienden består av flygenheter.
- Väntorder
  - Vänta på klartecken.
  - Slå läger.
- Trafikstockning, bör undvikas på planeringsstadiet.
  - Byt väg.
  - Fortsätt, det normala alternativet då radiotystnad råder.
  - Fråga befäl.
- Koordinering mellan grupper
  - Sköts av transportförband, implementeras ej i modellen utan lämnas till användaren.
- Ändrat hotläge
  - Byt formation (Avstånd, ordning, orientering etc.)
  - Avbryt marsch (Övergå till gruppering, läger etc.)
  - Byt väg.
  - Fortsätt.
  - Ändra hastighet.
- Utslaget/Immobiliserat fordon upptäcks
  - Identifiera fordonet.
  - Meddela befäl.
  - Ta skydd (Om hot föreligger).
  - Fortsätt.

Bilaga B. Modellbeskrivningar

FOIIF Ground Vehicle Commander

Description

The GroundVehicleCommander cognitive model simulates the behaviour of a military ground vehicle acting within a larger military unit. The model is generic in terms of its placement within the hierarchy, the same model can be used for both a battalion commander and the commander of a single vehicle. The unit performs assignments and gives orders to its subunits as defined in its script. Currently the only assignment available in the model is a march-assignment. The model organizes a formation of all vehicles in a specified unit, and performs a march to a specified location, using either default or user-specified parameters.

Features

- Builds and maintains a command structure for all units under its command.
- Performs marches specified in a script, ordered by a state system.
- Plans its path if no path is specified, and/or follows full or partial instructions in a route file.
- Watches for and detects threatening aircrafts.
- Sets up an encampment at its destination.
- Detaches and reattaches subunits for separate assignments.

Recorders

No recorders are used.

Connection to other models

Cognitive Models

Model Name	Required?	Description
SimpleRadioManager	Yes	To provide communication channels to other units.

Equipment and other models

Model Name	Required?	Description
FOIIFQPGroundVehicle	Yes	The GVC model uses this platform to be able to move along its selected path and get basic vehicle behaviour.
FQCSimpleRadio	Yes	To be able to communicate with other units using the GVC model.
FOIIFGVCEyes	Yes	To detect enemies, obstacles in the units path, destroyed vehicles and various events.

Scripts

Cognitive dictionary script

```
ASSIGN STATE CONTROL_MODE = CONTROL_CENTRALIZED;

/* Transmit message set. */
TRANSMIT "FOIIFMsgBeginMarch" AT BEGIN_MARCH;
TRANSMIT "FOIIFMsgBeginFollow" AT BEGIN_FOLLOW;
TRANSMIT "FOIIFMsgStartedMarch" AT STARTED_MARCH;
TRANSMIT "FOIIFMsgMarchOrder" AT MARCH_ORDER;
TRANSMIT "FOIIFMsgTargetReached" AT TARGET_REACHED;
TRANSMIT "FOIIFMsgGoodToGo" AT GOOD_TO_GO;
TRANSMIT "FOIIFMsgMissionComplete" AT MISSION_COMPLETE;
TRANSMIT "FOIIFMsgStartedFollow" AT STARTED_FOLLOW;
TRANSMIT "FOIIFMsgUpdateRoute" AT UPDATE_ROUTE;
TRANSMIT "FOIIFMsgPromotion" AT PROMOTION;
TRANSMIT "FOIIFMsgUnitLost" AT UNIT_LOST;

/* Receive message set. */
RECEIVE "FOIIFMsgMissionComplete" USING FOIIFGVCProcessMsgMissionComplete;
RECEIVE "FOIIFMsgGoodToGo" USING FOIIFGVCProcessMsgGoodToGo;
RECEIVE "FOIIFMsgTargetReached" USING FOIIFGVCProcessMsgTargetReached;
RECEIVE "FOIIFMsgBeginFollow" USING FOIIFGVCProcessMsgBeginFollow;
RECEIVE "FOIIFMsgBeginMarch" USING FOIIFGVCProcessMsgBeginMarch;
RECEIVE "FOIIFMsgStartedMarch" USING FOIIFGVCProcessMsgStartedMarch;
RECEIVE "FOIIFMsgMarchOrder" USING FOIIFGVCProcessMsgMarchOrder;
RECEIVE "FOIIFMsgStartedFollow" USING FOIIFGVCProcessMsgStartedFollow;
RECEIVE "FOIIFMsgUpdateRoute" USING FOIIFGVCProcessMsgUpdateRoute;
RECEIVE "FOIIFMsgPromotion" USING FOIIFGVCProcessMsgPromotion;
RECEIVE "FOIIFMsgUnitLost" USING FOIIFGVCProcessMsgUnitLost;

/* Radio setup*/
PERFORM STARTUP USING FOIIFSRMStartup
 PARAMETER(NAME="RADIO",
           NETWORK=1,
           CHANNEL=1);
PERFORM SETUP_RADIO USING FOIIFSRMSetupRadio;

/* Processing threads. */
PERFORM STARTUP USING FOIIFGVCStartup;
PERFORM INIT USING FOIIFGVCInit;
PERFORM ASSESS USING FOIIFGVCAssess;
PERFORM SET_MARCH USING FOIIFGVCSetMarch;
PERFORM MARCH USING FOIIFGVCMarch;
PERFORM FOLLOW USING FOIIFGVCFollow;
PERFORM UPDATE_UNIT USING FOIIFGVCUpdateMember;
PERFORM COMMAND_CHAIN USING FOIIFGVCAddCommandChain;
PERFORM DESTINATION_REACHED USING FOIIFGVCDestinationReached;
PERFORM SHOW_SYMBOL USING FOIIFGVCShowSymbol;
PERFORM HIDE_SYMBOL USING FOIIFGVCHideSymbol;
PERFORM SEND_MESSAGE USING FOIIFGVCSendMessage;

EMPLOY COMDEVICE "RadioNtoN" ALIAS "RADIO"
  PARAMETER ("NETWORK" = 1);
EMPLOY SENSOR "GVCEyes";
```

Example Unit script

```
INCLUDE "GVC";

/* Platform that uses FOIIFEQPGV */
EMPLOY PLATFORM "BMP2";
ASSIGN COMMANDER "BatC";
LOCATE AT 59.728188 19.056801;
START 0;
INITIATE ASSIGN_LEADER INPUT(LEADER="BatC");
INITIATE INIT
         INPUT(LEADER = "BatC",
         GROUP_NAME = "Ledgrp",
         COMMAND_NAME="Komp 1",
         RANK=2);
INITIATE ASSESS;
```

Process Methods

INIT (FOIFGVCInit)

Initialises the cognitive model and builds the command structure between the different units.

Inputs

Variable Name	Default Value	Description
LEADER	-	The name used by FLAMES for the leader of this unit
GROUP_NAME	-	The name of the smallest logical group this unit belongs to, typically the name of its platoon
COMMAND_NAME	-	The name of the units command
COMMAND_LEVEL	0	The hierarchy level the unit has, where 0 is the lowest

ADD_COMMAND_CHAIN

The function informs a unit of the current chain of command. Used internally only.

UPDATE_MEMBER

The function informs a leader of which units the sender commands. It is used internally to update the command structure.

ASSESS (FOIIFGVCAssess)

Regularly checks sensors and internally stored data to see if there are any events that the unit should respond to, or any actions to perform. It starts marches when all units are ready, checks for obstacles on the way and if there are nearby enemies. It also identifies destroyed vehicles and checks if they belong to its own forces, and tries to rebuild the hierarchy if the lost unit is under the detecting unit's command, otherwise it informs its leader about the dead unit.

DESTINATION_REACHED (FOIIFGVCDestinationReached)

A function called by the platform to inform the cognitive model that the destination of its current march has been reached. The function can generate `TARGET_REACHED` and/or `MISSION_COMPLETE` if the requirements are met.

MARCH (FOIIFGVCMarch)

Makes the unit start moving along a route it has previously received or constructed. MARCH runs continuously until the destination of the march is reached, or the march is aborted. This function should never be called directly. Instead `SET_MARCH` should be called.

FOLLOW (FOIIFGVCFollow)

Same as MARCH, except that the unit will follow another unit that is moving along the same path at a given distance. If the unit that is being followed dies, follow terminates and starts MARCH instead.

SET_DEFAULT_FORMATION (FOIIFGVCSetDefaultFormation)

This function changes the default formation to be used while marching. Valid formations are "COLUMN_MEKBAT" and "COLUMN".

Inputs

Variable Name	Value Identifier	Description
FORMATION	Name	Name of the formation to use.

SHOW_SYMBOL (FOIIFGVCShowSymbol)

The function shows a military symbol of the unit's current command at the geometrical centre of the unit and its subcommanders.

HIDE_SYMBOL (FOIIFGVCHideSymbol)

The function hides a symbol that has previously been displayed by the use of SHOW_SYMBOL. If the symbol is not already shown the function does nothing.

SEND_MESSAGE

Wrapper function for messages, used to handle the case when units are not responding to messages.

SET_MARCH (FOIIFGVCSetMarch)

The function initiates a march to be performed by either this unit or one of its subordinates, specified by RECEIVER. The march is not performed if the unit receiving it is already performing a blocking mission. If the march requires a state to be set before being initiated the function is suspended until this state is set. If a routefile is specified, the unit will follow the instructions in this (see below), and fill in the blanks using a shortest path algorithm.

Inputs

Variable Name	Value Identifier	Description
DESTINATION_LAT	Real	Latitude position of destination.
DESTINATION_LON	Real	Longitude position of destination.

Parameters

Parameter Name	Default Value	Description
SPEED	DEFAULT_MARCH_SPEED	The speed the march formation should try to keep.
ROUTEFILE	-	A file specifying a full or partial route to the destination.
RECEIVER	-	A unit under this units command that should receive the order
FORMATION	-	A formation to use while marching
DETACHED	0 (false)	If true the unit will not participate in any other missions before this one is finished.
VEHICLE_DISTANCE	DEFAULT_MARCH_DISTANCE	Standard distance between vehicles in the march.
WHEN_STATE	-	A state which must have been set before this march is performed
REQUIRED_BY_STATE	-	A state which can only be set when this march is complete.

Received Messages

FOIIFMsgBeginFollow

Message that orders the receiving unit to start following a specified unit. The message works recursively and the receiver sends all units under its command that is not detached on this march. The function does nothing if the receiver has not already received the correct march order for this march, or if it has already started marching.

FOIIFMsgBeginMarch

Message that orders the receiver to start marching, it works the same way as FOIIFMsgBeginFollow.

FOIIFMsgGoodToGo

Reports to leader that the unit is ready to perform a specified mission. The marchleader will await this message from all its subordinates before initiating the march.

FOIIFMsgMarchOrder

Information about or order for a march directed to the receiver or one of its subunits. This message needs to be sent to all participants before the march can be actually started by sending FOIIFMsgBeginMarch and FOIIFMsgBeginFollow messages to the units that should participate in this march.

FOIIFMsgMissionComplete

Informs receiver that the sender has finished the mission specified in the message. If the mission was performed detached from the rest of the force, the leader who receives it reattaches the unit who has performed the mission, and the sending unit is now ready to perform new missions.

FOIIFMsgStartedFollow

Notifies the receiver that the sender has started following the receiver.

FOIIFMsgTargetReached

The unit informs the leader that it and all of its subunits have reached the destination of the current mission. When all units under the command of the receiver have sent this message, and the receiver itself has reached the target, it will consider this march complete.

FOIIFMsgPromotion

Informs a unit that it has now been promoted and which units it is now leading.

FOIIFMsgUnitLost

A message informing the receiver that a unit has been lost.

Internal methods

This chapter describes a number of internal functions that are not callable from the script, and when they are used.

prepareMarch

A function that "fills in the blanks" in a call to SET_MARCH or a received marchorder. Only used by the marchleader.

buildMarchFormation

BuildMarchFormation constructs a data structure storing the positions of each subunit participating in the march. Used by the marchleader when preparing a march and by subunits when they receive the actual order to start the march. See also SendNext.

sendNext

Sendnext is called when the next unit in the marchformation is about to be sent. This is performed by ASSESS on the start of the march and when a started march message is received for the following units. The function uses the data structure created by buildMarchFormation which should be executed before any calls to sendNext.

propagateMarchOrder

The function sends a complete marchorder to all subordinates that should participate in the current march (everyone that is not detached). It also calls buildEncampment to give each subunit a place to position itself at the destination.

calculateTimeToMove

The function checks if and when the current unit should start moving to make it to the formationpoint in time for the march. If the unit is close enough to the formationpoint it simply generates a goodToGo message.

predictCollision

The function is called by MARCH and FOLLOW and tests if there are any other units which will cross the current unit's path in the near future. If there are any such units, it returns a suggested speed modification to avoid collision. It is primarily supposed to handle meetings at intersections and does not handle frontal collisions and overtaking.

inside

The function is called by predictCollision and performs a test if two boundingboxes intersect.

replaceUnit

replaceUnit attempts to replace a dead or lost subunit in the internal hierarchy. The replacer will be the subunit of the lost unit that has the highest rank. If two units exist who have the same rank, the one with the lowest ID will be selected. If no subunits exist, the lost unit will simply be deleted. The function also calls recoverMissions to try to make sure that any missions currently performed by the lost unit are not lost.

recoverMissions

Tries to perform the current mission even if key units have been lost and resends important messages.

Internal classes

Route

This class is used to build and store routes. It stores three routes for each march, between the start point and the formation point, between the formation point and the destination and between the destination and the position at the encampment when the march is complete. The route contains a node map that is common for all route objects.

PUBLIC MEMBER FUNCTIONS

buildNodeMap

This function creates a node map if it does not exist.

buildRoute

If a route file is specified, this function uses readRoute to build a route, otherwise it uses calculateRoute in NodeMap.

buildRouteFromDestination

This function uses calculateRoute in NodeMap to build a route between the destination and this vehicles position at the encampment.

buildStartRoute

This function uses calculateRoute in NodeMap to build a route between the start point and the formation point.

dropRoadSegment

This function uses dropRoadSegment in NodeMap to remove the segment between two points of a road.

getDestination

Returns the destination.

getFormationPoint

Returns the formation point.

getIndexInRoute

This function returns the first index to a node in the route that matches a specified coordinate or -1 if no node matches.

getRoadSegment

This function picks one vertex on each side of a given point on a specified road.

getRouteCopy

Returns a Model Argument List containing the main route.

getRouteFromDestinationCopy

Returns a Model Argument List containing the route from the destination.

getRouteToFormationPointCopy

Returns a Model Argument List containing the route to the formation point.

rebuildRouteFromIndex

Recalculates the route from a given index.

restoreRoadSegment

Restores a segment of a road that has been removed by dropRoadSegment.

setDestination

Sets the destination.

setFormationPoint

Sets the formation point.

setRoute

Sets a new route and checks that it is valid.

setRouteFile

Stores the filename of a route file.

PROTECTED MEMBER FUNCTIONS

getMALfromVector

This is an internal function that creates a Model Argument List (MAL) from a route stored in a STL vector.

pruneRoute

This is an internal function that removes nodes from a route that have the same coordinate as the previous node.

readCoordinate

This is an internal function that reads a single coordinate from the route file.

readRoute

This is an internal function that parses a route file and uses the node map to create a main route.

PROTECTED MEMBERS

mRouteVector

The route between the formation point and the destination.

mStartRouteVector

The route between the initial location and the formation point.

mEndRouteVector

The route between the destination and the final location.

mFormationPoint

The formation point.

mDestination

The destination.

mRouteFilename

The name of the route file.

mNodeMap

The node map, which is shared by all units.

NodeMap

NodeMap handles roads and route calculations. All terrain calculations should be inserted into this class. It might be easier to rewrite the whole class when a suitable terrain API is available because a general terrain algorithm could also include the roads.

PUBLIC MEMBER FUNCTIONS

buildRoads

This function collects road vertices, intersections and road names from FLAMES and stores in STL structures in this class. A graph with nodes at the ends of the roads and at intersections between roads is built.

buildTerrain

This function is not implemented yet. It is intended to construct a node map from the terrain.

calculateRoute

Calculates a route on the road network between two coordinates and optionally starts on a given road. It uses createNodeOnRoad to create a node at the point on the road network closest to a given start point and a node closest to a given end point. Then if the sum of the distances to and from the road network is shorter than the distance between the start point and the end point, it uses shortestPath to build the path. The temporary nodes are removed from the node map after the route is calculated.

calculateRouteToIntersectingRoad

Calculates a route from a start point to a specified road. If a first road is specified, this road is followed to an intersection with the new road. If nearIntersection is set, the intersection closest to this coordinate is used, otherwise shortestPathToRoad is used to find the shortest path to the closest intersection on the new road. Creation and deletion of temporary nodes is the same as for the function above.

dropRoadSegment

Removes any edges on a specified road between two specified points or any edges starting or ending on the road segment. No nodes are removed. The removed edges are stored so that the segment can be restored later.

getRoadSegment

Picks a vertex on each side of a specified point from the road farther away than a specified distance.

restoreRoadSegment

Restores all edges for a specified road segment that have been removed by a call to dropRoadSegment.

PROTECTED MEMBER FUNCTIONS

addEdge

Inserts an edge between two nodes.

addJunction

Inserts a new node and edges to and from the previous node on that road.

addNode

Inserts a new node at a given position if there is no other node at that position.

combineNodes

Adds edges between two nodes if they only have edges to and from the same nodes.

createNodeOnRoad

Inserts a node at the position on the roads that is closest to a given position if there does not already exists a node on that position. Edges are added to and from the closest nodes on the road where the new node is inserted.

getClosestPositionOnRoad

Returns the point on a road that is closest to a specified point.

getNodeIndex

Returns the index of the first node in the node vector with a position that is equal to a specified position.

getRoadIndexBetweenNodes

Returns the index of the road between two specified nodes.

getVertices

Stores road vertices in a vector between two points on a specified road.

popLastNode

Removes the last node in the node vector and any edges pointing to it.

removeEdge

Removes the edge between two specified nodes.

shortestPath

Calculates the shortest path on the road network from a start node to an end node with Dijkstras algorithm.

shortestPathToRoad

Calculates the shortest path on the road network from a start node to an intersection on a specified road that could be either closest to the start point or a specified point.

PROTECTED MEMBERS

mRoadNodeVec

The nodes in the road map.

mRoadVec

A 2D vector containing the vertices of all roads.

mRoadNameVec

The road names in a vector.

mRoadNameMap

The road names in a map.

mRemovedSegments

Removed road segments that can be restored.

mIntersectionVec

The intersections.

UnitTree

This class is used to build a hierarchy structure in each unit, describing the unit and the units it commands. It is a tree structure where the own unit is the top node, and subordinates directly commanded by it is its children. The tree can contain an unspecified number of levels and units.

resetUnitGoodToGo

Resets the goodToGo flag for all units in this structure.

isUnitGoodToGo

Checks if everyone is ready to march.

setUnitOnMission

Sets the mission flag that declares that this unit is currently performing a mission.

isUnitOnMission

Returns true if this unit or any of its non-detached subordinates is performing a mission.

getHighestRankingSubordinate

Gets the subordinate with the highest rank in the tree. If there are two units with the same rank, the one with the lowest ID is selected.

getUnitByID

Returns a unit by it’s ID.

getUnitByType

Returns the first unit of the corresponding type and size.

getUnitByName

Returns a unit by it’s name.

getCommander

Returns the commander of a specified unit.

getFMALCopy

Creates a FMAL object containing this unitTree.

addSubUnit

Adds a new subordinate of the current unit.

addLevelCommander

Adds a commander at the same level as the current unit.

dropSubUnit

Removes the specified subunit from the tree.

dropLevelCommander

Removes the specified commander from the tree.

replaceSubunit

Replaces a subunit with a new one.

calculateNrOfUnits

Counts the units in the tree and their vehicle types.

clearReservations

Removes all reservations that are set by mission functions.

FOIIFQPGroundVehicle

Description

FOIIFGroundVehicle is an equipment model of a ground vehicle. It simulates the movement along a route at a desired speed.

Features

- Reduces speed before turns and before it stops at the destination.
- Generates a "Destination reached" message when the main route is complete.
- Allows added route data and jumping in routes during a march, which is needed to avoid obstacles.
- Can return the distance from the current location to a specified node in the route, which is needed by the follow functionality.

Recorders

No recorders are used.

Connection to other models

Cognitive Models

Model Name	Required?	Description
FOIIFGroundVehicle Commander	No	The FOIIFGroundVehicleCommander is intended to control this equipment model by providing a route and a desired speed to follow.

Process Methods

ClassInitialize(void)

Defines parameters that can be altered in FORGE.

Control(FMALObj MAL)

Allows a cognitive model to control the equipment model with a Model Argument List (MAL) containing one or more of the following input parameters by using FEQuipmentControl.

Inputs

Variable Name	Value Identifier	Description
ROUTE	MAL	A linked list of MAL objects, where each object contains a node in the route.
FINALROUTE	MAL	Same as above but describes the route between the destination and the units position at the encampment.
SPEED	REAL	Desired speed.
INDEX	INTEGER	Sets index to the next node in the main route.
FINALHEADING	REAL	The vehicles heading angle in radians from north at the encampment.

Move(FJulianType Time)

Moves the vehicle along the route a distance depending on current speed, current acceleration and the time since last update. A DESTINATION_REACHED message is generated when the main route is complete. setSpeed is used to update current speed and current acceleration.

Inputs

Variable Name	Description
Time	Current time in simulation, used to calculate the time since last update.

Query(FMALObj MAL)

Allows a cognitive model to retrieve data from this model by using FEntityQuery.

Inputs

Variable Name	Value Identifier	Description
TYPE	Int	The type of vehicle.
NEXT_NODE_POSITION	FVectorType	The ECR coordinate of the next node in the route.
NEXT_NODE_INDEX	Int	Index to the next node in the route.
DISTANCE_TO_NEXT_NODE	double	The distance to the next node.
DISTANCE_TO_NODE	double	The distance to a node with index specified by an integer value provided by this parameter.

Internal Methods

appendRoute(FMALObj route, bool inMainRoute)

Appends the route to any previously stored route.

Inputs

Variable Name	Description
Route	A linked list containing a route.
inMainRoute	False if the route is between the destination and the encampment, true otherwise.

placeOnGround(void)

Places the vehicle on the terrain at its current location.

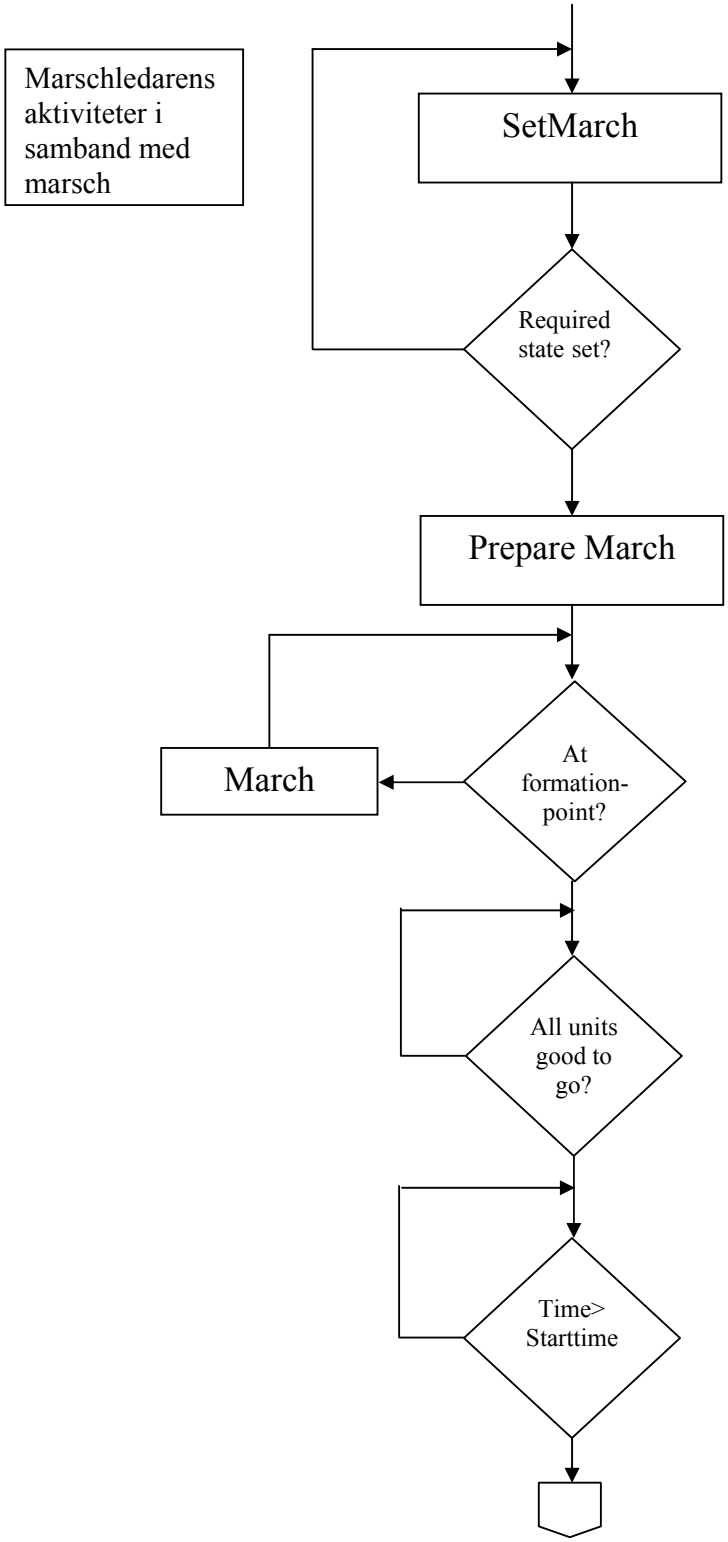
setSpeed(FJulianType deltaTime)

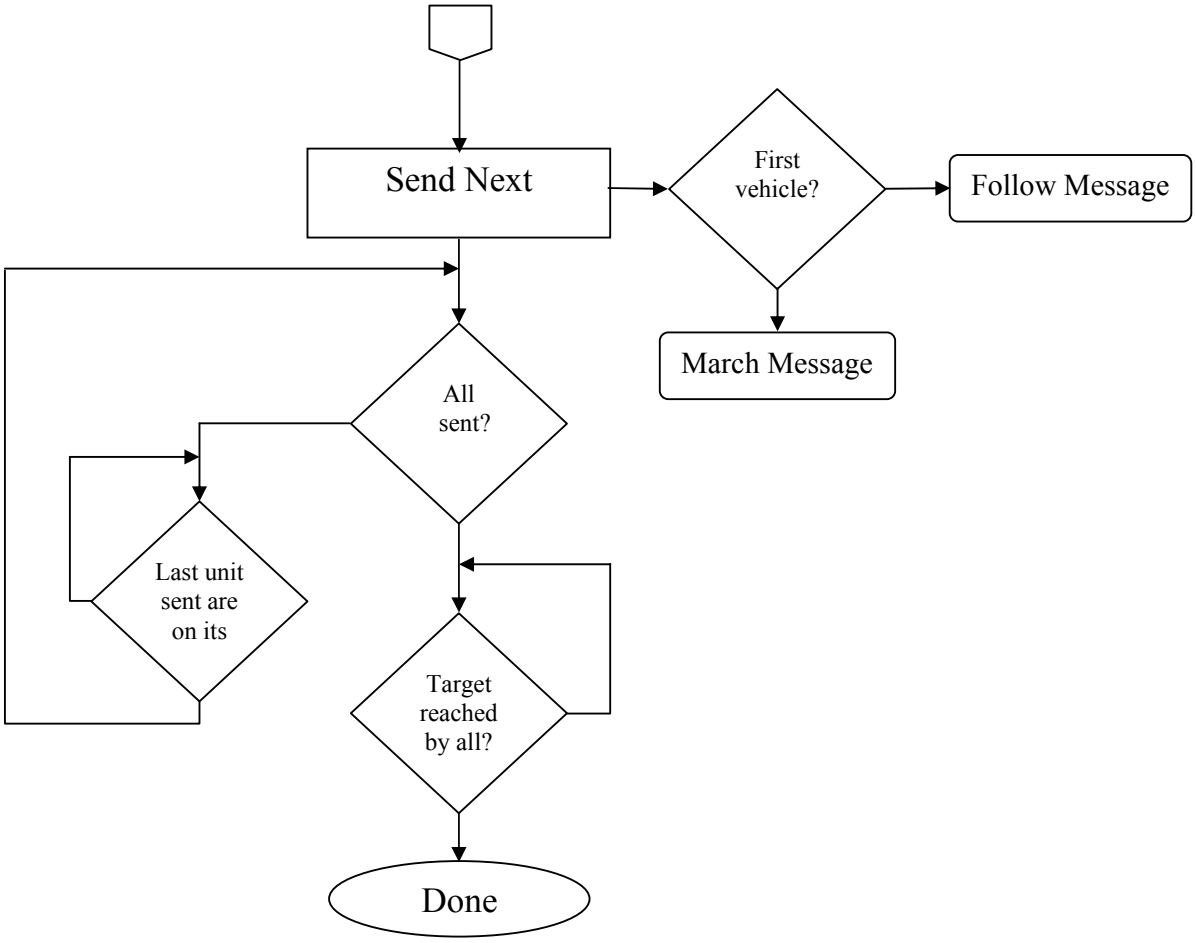
Adjusts current speed according to current acceleration, time since last update and max speed.
Calculates if it is necessary to reduce the current speed before a turn or the destination, and in that case recalculates the current acceleration so that the vehicle stops at the destination or pass a turn at a low enough speed. Otherwise current acceleration is set in the interval –max deceleration to max acceleration to try to mach the desired speed

Inputs

Variable Name	Description
deltaTime	Simulation time since last update.

Bilaga C. Flödesschema




Send Next
First vehicle?
Follow Message
March Message
All sent?
Last unit sent are on its
Target reached by all?
Done