FFI RAPPORT

SIMBA - DOKUMENTASJON AV KILDEKODE

HALSØR Marius, EIDE Morten

FFI/RAPPORT-2003/01711

FFI-III/798/139

SIMBA - DOKUMENTASJON AV KILDEKODE

HALSØR Marius, EIDE Morten

FFI/RAPPORT-2003/01711

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
Postboks 25, 2027 Kjeller, Norge

FORSVARETS FORSKNINGSINSTITUTT (FFI)
Norwegian Defence Research Establishment

P O BOX 25
NO-2027 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER	2) SECURITY CLASSIFICATION	3) NUMBER OF PAGES
FFI/RAPPORT-2003/01711	UNCLASSIFIED	
1a) PROJECT REFERENCE	**2a) DECLASSIFICATION/DOWNGRADING SCHEDULE**	102
FFI-III/798/139	-	

4) TITLE

SIMBA - DOKUMENTASJON AV KILDEKODE

SIMBA - DOCUMENTATION OF SOURCE CODE

5) NAMES OF AUTHOR(S) IN FULL (surname first)

HALSØR Marius, EIDE Morten

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)

7) INDEXING TERMS

IN ENGLISH:	IN NORWEGIAN:
a) Simulation program	a) Simuleringsprogram
b) Tank	b) Stridsvogn
c) Anti-tank	c) Panserbekjempelse
d) Source code	d) Kildekode
e)	e)

THESAURUS REFERENCE:

8) ABSTRACT

During FFI-project 701, “Fremtidige panserbekjempelsesvåpen” (Future anti-tank weapons), a simulation program for tank/anti-tank warfare named SIMBA was developed. The program was meant to handle combat on maximum battalion level. Each tank and anti-tank system is modeled as a single unit. In addition to Tanks and anti-tank weapons, the model can also handle use of sensor fused and conventional artillery.

The simulation model actually consists of four different programs. One is a preprosessor, in which the scenarios are specified. This is where one tells SIMBA what the initial situation looks like, and what the orders are for each separate unit. Then there is a simulation program, which does the actual simulation. Finally, there are two programs for post-processing, one for studying a single simulation in detail, and one for examination of the statistics from many simulations. The model is stochastic, so each simulation could yield a different result.

This document describes the source code for these four programs. It is aimed at those who wish to make changes in the codes and improve the model. Those who only need to know what the model does, and how to operate it, are referred to the user manual, (1).

9) DATE	AUTHORIZED BY This page only	POSITION
2004-05-04	Johnny Bardal	Director

ISBN-82-464-0863-1

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (when data entered)

INNHOLD

SIMBA - DOKUMENTASJON AV KILDEKODE

1 INNLEDNING

SIMBA (SImuleringsModell på BAtaljonsnivå) ble utviklet under prosjekt 701, Fremtidige Panserbekjempelsesvåpen. Programmet var ment som et verktøy for å støtte prosjektet med kvantifisering av stridsutfall i strider der stridsvogner (Strv) og stormpanservogner (SPV) var i trefning med panserbekjempelsesvåpen (PB-våpen). To av områdene der SIMBA har blitt anvendt, er for å finne forventet sårbarhet til PB-lag (for bestemmelse av forventet antall avfyrte missiler pr. system), og til å finne en balanse mellom direkteskytende og indirekteskytende (smart) PB. De stridskomponentene som SIMBA takler, er Strv/SPV, PB, smart- og konvensjonelt artilleri. Programmet er laget på en slik måte at det skal være enkelt å legge til nye komponenter etter behov.

SIMBA består av 4 forskjellige programmer: Preprosessor, Simulator, Postprosessor og Kartlogg. SIMBA Preprosessor er et program for generering av scenarier. Det er utviklet i MatLab under UNIX, og er senere tilpasset MatLab for MS Windows (V.6.1 R12.1). SIMBA Simulator tar scenariet generert under preprosessoren som input, og foretar selve simuleringen av scenariet et antall ganger bestemt av brukeren. Som output fra simuleringsdelen kommer en mengde tekstfiler (loggfiler), én for hver simulering. Dette programmet ble opprinnelig skrevet i SoftBench under UNIX men også dette er senere tilpasset MS Windows (Visual C++ v.6.0). SIMBA Postprosessor er et program for å trekke ut relevant informasjon fra loggfilene. Det tar et brukerbestemt antall loggfiler som input, og regner ut gjennomsnittsverdier og standardavvik til aktuelle størrelser (sårbarhet, effektivitet, etc.). SIMBA Kartlogg viser hendelsesforløpet til en enkelt simulering grafisk. Den tar en enkelt loggfil som input, og viser utviklingen i scenariet steg for steg. Kartlogg og Postprosessor er utviklet i Visual C++ (v.6.0) for MS Windows.

De forskjellige programmene har blitt utviklet i flere etapper, og det har vært flere programmerere med underveis. Mange av disse er ikke lenger ved FFI, og det har derfor ikke alltid vært lett å dokumentere alle deler av programmet.

Dette dokumentet beskriver kildekoden til de 4 programmene som utgjør SIMBA, og gir en beskrivelse av utseendet til loggfilene som genereres av simuleringsdelen. Dokumentet er således myntet på personer som ønsker å gjøre endringer i programmet, enten ved å legge til nye elementer eller endre funksjonalitet. For de som bare skal bruke programmet og ikke har behov for å gjøre endringer i kildekoden, henvises det til (1).

SIMBA videreutvikles etter behov. Dette dokumentet beskriver SIMBA slik det var ved utgangen av år 2003. Gjeldende versjonsnumre på dette tidspunktet er:

SIMBA Preprosessor	: v.2.5
SIMBA Simulator	: v.2.0
SIMBA Postprosessor	: v.2.1
SIMBA Kartlogg	: v.4.1

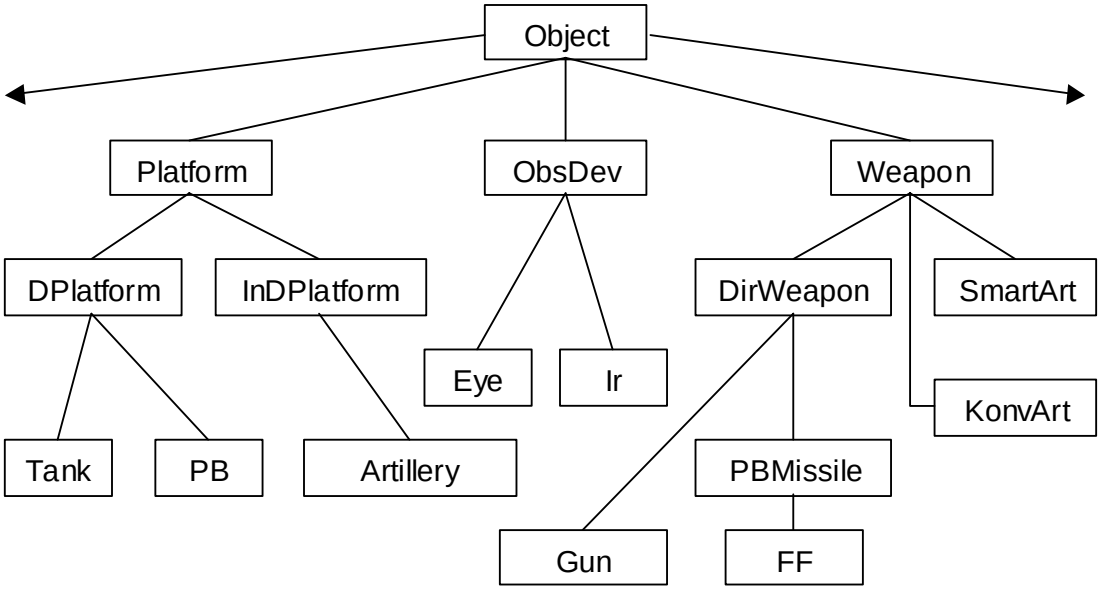
2 SIMBA SIMULATOR

2.1 Oversikt

SIMBA leser inn data fra tekstfiler generert i SIMBA Preprosessor, og simulerer en hendelsesutvikling for et scenario. Utviklingen oppdateres for hvert tidssteg, der tidssteget er en forhåndsbestemt størrelse satt av brukeren. Dette fortsetter til et avbruddskriterie er oppfylt. Enkelte hendelser styres ved Monte Carlo simulering, så det samme scenariet kan få forskjellige hendelsesforløp ved to forskjellige simuleringer.

SIMBA Simulator ble opprinnelig utviklet i C++ under SoftBench i UNIX, men er siden flyttet over til Visual C++ for MS Windows. I dette kapitlet beskrives hvordan de forskjellige klassene arver egenskaper av hverandre. Beskrivelsen av hvilke egenskaper som ligger på de forskjellige nivåene, kommer i neste kapittel.

Begrunnelsen for klassen Object er at det er laget en listestruktur (ObList) som tar elementer av denne klassen. Dermed kan den samme listestrukturen brukes til alle elementer, istedenfor at man har én listestruktur for plattformer, én for siktemidler osv. Det finnes flere klasser enn de opptegnede som er underklasser av Object og som også benytter samme listestruktur, f.eks. Flag og Missile.



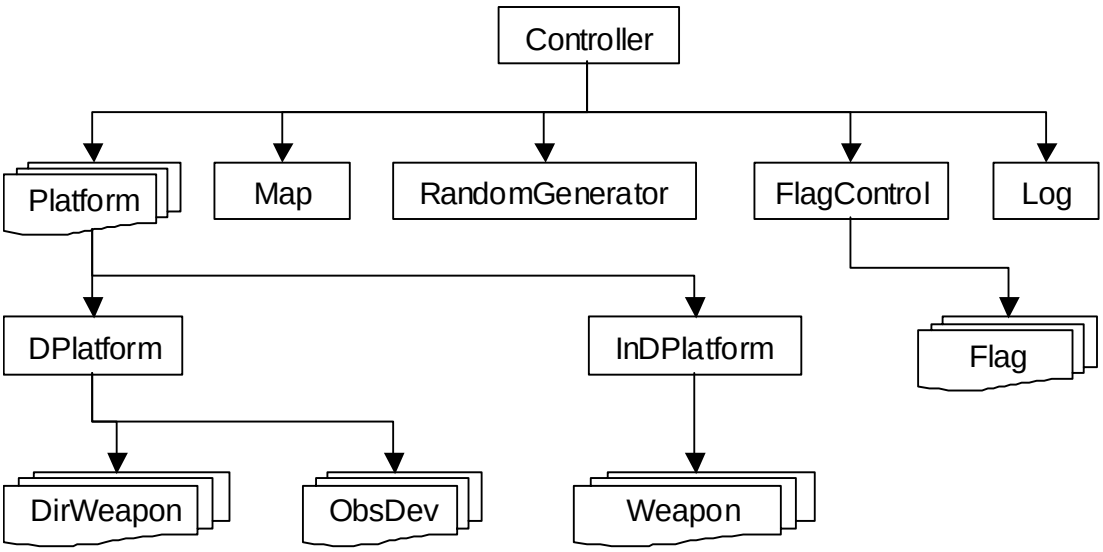
Figur 2.1: Utdrag av klassestruktur

Det finnes flere klasser enn det som fremkommer av figuren, men disse klassene er alle instanserbare, og har ingen superklasser (med unntak av de som har Object som superklasse). Eksempler på slike klasser er Controller og Flag.

I resten av dette kapitlet beskrives klassene, samt deres metoder og viktigste medlemmer. Metoder som er rent virtuelle (pure virtual) er ikke beskrevet; de beskrives i underklassene, der de ikke lenger er virtuelle.

2.2 Controller

Ved kjøring av SIMBA Simulator opprettes det én instans av klassen Controller. Denne sørger for innlesing av alle datafilene som inngår i simuleringene og opprettelse av de nødvendige objektene. Den repeterer simuleringen et gitt antall ganger, oppdaterer hendelsesutviklingen hvert tidssteg og avbryter simuleringen etter visse kriterier. Objektstrukturen for Controller er vist i figur 1.2. De objektene som her er markert med flere bokser, representerer en liste av objekter, for eksempel styrer objektet FlagControl en liste med objekter av typen Flag.



Figur 2.2: Oversikt over Controller-styrte enheter

2.2.1 Metoder i *Controller*:

- **RunScenario**
  Kaller metoden Readscenario og sørger for at simuleringen repeteres et gitt antall ganger. Metoden inneholder algoritmen som styrer hendelsesutviklingen til objekter av typen Platform.

- **ReadScenario**
  Leser inn scenariofila og sørger for innlesning av de andre datafilene, og oppretter alle objekter som skal inngå i simuleringen. Disse objektene leser selv inn fra fil de data de måtte trenge.

- **ReadMap**
  Sørger for at objektet Map leser inn terrengdata, samt eventuelle vegetasjonsdata og kunstige ”vegger” av vegetasjon.

- **ReadPlatforms**
  Leser inn data fra plattformfila og oppretter Platform-objektene, samt kaller metoden ReadPlatformData i det nyopprettede Platform-objektet.

- **ReadCommands**
  Leser inn data fra kommandofila (dreieboka), og kaller metoden ReadCommands i det aktuelle Platform-objektet, som leser videre fra samme fil.

- **GetPlatform**
  Finner en plattform med et bestemt ID-nummer.

- **CheckStatusOnScenario**
  Sjekker om avbruddskriteriene er oppfylt.

2.3 Map

Det blir opprettet én instans av klassen Map. Den håndterer alle rutiner som har med kartet å gjøre. Kartet er beskrevet av fire filer:

- **<kartområde>.dat** inneholder generell informasjon om kartutsnittet, som koordinatene til sørvestlige hjørne og utstrekning, samt oppløsning i øst-vest og nord-sør retning.
- **<kartområde>.ter** inneholder høydeinformasjon for kartutsnittet (høyden til terrenget, ikke vegetasjonen). Høydedata er representert som en matrise med diskrete punkter, *nx* i øst-vest retning ganger *ny* i nord-sør retning. Avstanden mellom punktene er *dx* meter i øst-vest retning og *dy* meter i nord-sør retning. Høyden er gitt i desimeter.
- **<kartområde>.veg** er på samme form som .ter-filen, og beskriver høyden på vegetasjonen i området. En forskjell fra .ter-filen er at denne beskriver høyden i rektanglet mellom punktene som brukes i .ter-fila. Det er derfor (nx-1)(ny-1) vegetasjonspunkter. Vi har ikke tilgang til gode digitale kart for vegetasjonshøyde, men når man får tilgang til det, er SIMBA klar til å håndtere det. SIMBA kjøres derfor som regel med denne fila tom. Dette gir en feilmelding uten å avbryte programmet, og programmet fungerer fint uten denne fila.
- **<kartområde>.wall** brukes når man ikke har tilgang til digitale kart som inneholder informasjon om høyden på vegetasjonen. Fila beskriver ”vegger” av vegetasjon med en gitt høyde, startpunkt og sluttpunkt. Disse veggene er ment å skulle ta hensyn til de viktigste egenskapene til vegetasjonen i området, ved å hindre fri sikt mellom to posisjoner som uten vegetasjon ville hatt fri sikt, men som gjennom befaring er påvist å ikke ha det.

Metoder

- **ReadDataFiles**
  Besørger innlesningen av .dat-, .ter-, .veg- og .wallfilene.

- **GetNx, GetNy, GetDx, GetDy**
  Disse metodene, og alle andre prosedyrer som har navn som begynner med *Get*, returnerer verdien til den påfølgende variabelen.

- **FreeLine**
  Sjekker om det er fri sikt mellom to punkter. For å avgjøre dette kalles metoden FreeThrSquare for hvert rektangel mellom start- og sluttpunkt.

- **TerrainHeight**
  Finner høyden i et punkt. Dersom punktet ligger mellom datapunktene i .ter-fila, brukes interpolasjon.

- **VegetationHeight**
  Som *TerrainHeight*, bortsett fra at det her er høyden på vegetasjonen som kalkuleres.

- **AddSmoke**
  Denne metoden benyttes ikke i nåværende versjon av SIMBA, men kan eventuelt tas i bruk senere (det er bare å fjerne noen bortkommenteringer i metoden Defend i objektet TANK). Formålet med metoden er å simulere utlegging av beskyttelsesrøyk. Dette gjøres ved å plassere et ugjennomsiktig kvadrat med sidekanter *smokesize* rundt den utleggende enheten. Til plassering av dette kvadratet kalles metoden *PlaceWall* 4 ganger.

- **RemoveSmoke**
  Denne metoden er heller ikke i bruk, men må brukes dersom metoden over skal brukes. *RemoveSmoke* fjerner røyk, og det gjøres ved å legge et nytt kvadrat oppå kvadratet som representerer røyken, med like stor men negativ høyde. Ettersom høyden til to ”vegger” legges sammen, tilsvarer dette å fjerne hindringene.

- **ReadFileDat, ReadFileTerrain, ReadFileVegetation, ReadWallFile**
  Åpner og leser den aktuelle filen (.dat, .ter osv.).

- **FreeThrSquare**
  Sjekker om det er fri sikt gjennom en ”rute”. En rute er i denne sammenhengen området som ligger mellom 4 punkter i .ter-filen.

- **PlaceWall**
  Leser .wall-filen og plasserer vegetasjonsveggen i terrenget. Kan også brukes i forbindelse med beskyttelsesrøyk.

2.4 Platform

Hver enhet som deltar i striden opprettes som én instans av klassen Platform. Klassen har to underklasser; Dplatform, som beskriver direkteskytende plattformer som f. eks. stridsvogner, og InDPlatform, som beskriver indirekteskytende plattformer som f. eks. artilleri. Hver

plattform har et ID-nummer som er unikt, og et navn (f.eks ”T80”) som bestemmer beskyttelsen til plattformen. Utover dette er en liste over mulige mål blant de viktigste egenskapene til dette objektet.

2.4.1 Metoder

- **ReadCommands**
  Leser kommandoene fra kommandofila og legger dem i en kommandoliste.
- **GetNextCommands**
  Henter den neste ordren fra kommandolista og setter denne ordren som aktiv.
- **AddShot**
  Inkrementerer variabelen Shots, som forteller hvor mange skudd som er avfyrt.
- **SetSmoke**
  Setter variabelen Smoke lik TRUE, og setter tiden den skal forbli TRUE til et antall sekunder gitt av den globale variabelen TimeOfSmoke. Dette beskriver at plattformen har avgitt skuddsignatur, og hvor lenge skuddsignaturen kan detekteres.
- **SetHitStatus**
  Denne metoden kalles når plattformen er truffet. Plattformen får en gitt status bestemt av treffet. Var det f.eks. en mobility-kill, får plattformen status ”mobility-killed” (Mkilled). Hvis den allerede hadde den skaden som den ble påført, registreres det som treff uten virkning (”hit”). Hvis den allerede hadde en annen skade, slik at det nye treffet totalt ødela plattformen, får den status ”Total-killed” (Tkilled), som betyr at plattformen er fullstendig ødelagt, og den deltar ikke lenger i simuleringen.
- **InTargetList**
  Sjekker om en plattform ligger i target-listen. Returnerer NULL dersom den ikke gjør det, eller target-objektet som peker til plattformen dersom den gjør det.
- **AddTarget**
  Legger til et nytt mål i target-listen.
- **RemoveTarget**
  Fjerner et mål fra target-listen.
- **InScanList**
  Sjekker om en plattform ligger i scan-listen. Scan-listen inneholder alle de plattformer som ligger innenfor siktefeltet, og som det er en klar siktelinje til (mao. alle mål det er mulig å detektere).
- **AddScan**
  Legger til en plattform i scan-listen.
- **RemoveScan**
  Fjerner en plattform fra scan-listen.

I tillegg kommer en del metoder av typen **Get*Variable***, som returnerer verdien til *Variable*.

2.5 Target

Når en plattform (A) kan beskytes av en annen plattform (B), legges A i target-listen til B. Når den legges i targetlisten, legges den der som et Target-objekt, ikke som et Platform-objekt. Det er altså snakk om den samme enheten, men klassen Target har noen andre egenskaper enn klassen Platform. Alle Target-objekter inneholder en peker til det tilhørende Platformobjektet, i tillegg til informasjon om plattformens status som mål.

2.5.1 Metoder

- **Identified**
  Dersom det har gått en forhåndsbestemt tid siden målet ble detektert, setter denne metoden medlemmet *identified* (samme navn, men liten i) til TRUE, og målet anses som identifisert som en fiende.

- **SetLastDetectionTime**
  Så lenge målet kan sees, er LastDetectionTime lik tiden i øyeblikket. SetLastDetectionTime kalles hvert tidssteg for alle *Target* som kan sees, og oppdaterer denne tiden.

- **AddShot**
  Inkrementerer variabelen noOfShots, som forteller hvor mange skudd som er skutt mot dette Targetobjektet.

- **SetShotAt**
  Kalles dersom målet er beskutt med maks antall tillatte skudd, og setter variabelen shotAt lik TRUE.

- **IsShotAt**
  Returnerer verdien av variabelen shotAt.

I tillegg kommer en del metoder på formen **Get*Variable***, som returnerer verdien til *Variable*.

2.5.2 Viktige medlemmer av klassen Target

- **target**
  Dette er en peker til det tilhørende Platformobjektet.

- **noOfShots**
  Forteller hvor mange skudd som er skutt mot dette Targetobjektet.

- **shotAt**
  Boolsk variabel som er TRUE dersom maks antall skudd er avfyrt mot target-objektet.

- **obsdev**
  Siktemiddelet som oppdaget target-objektet.

- **detectionTime**
  Tidspunktet da Targetobjektet først ble oppdaget.

- **lastDetectionTime**
  Siste tidspunkt da Targetobjektet kunne observeres.
- **IdentificationTimeDelay**
  Tiden det tar fra deteksjon til identifikasjon.
- **identified**
  Boolsk variabel som er TRUE dersom Targetobjektet er identifisert.

2.6 InDPlatform

InDPlatform beskriver de indirekteskytende plattformene. Dette er en virtuell klasse som foreløpig ikke inneholder noe særlig funksjonalitet ettersom det kun finnes en underklasse.

2.7 Artillery

Dette er en instanserbar klasse, med superklasse InDPlatform. InDPlatformer har ingen bestemt posisjon, men siden egenskapen posisjon ligger under superklassen Platform, får InDPlatform posisjonen 0:0. Posisjon for InDPlatformer er ikke nødvendig i SIMBA, da de ikke kan bli beskutt. En InDPlatform beskrives derfor av sitt siktepunkt og hvert våpens ’offset’ fra dette, altså området den har virkning i. Skuddrate, siktepunkt og bomavstand er sentrale parametre. Slik det er implementert i SIMBA Simulator, kan ett skudd slå ut flere plattformer hvis granatene har generell områdevirkning, slik konvesjonelt artilleri har. Smart artilleri søker utenfra og inn mot senter av området det kan se og velger det første målet som detekteres og som anses å være verdt å skyte på .

Metoder

- **ReadPlatformData**
  Leser inn data fra plattformfila.
- **Update**
  Oppdaterer plattformen hvert tidssteg. For Artilleri betyr dette i praksis kun å lagre plattformlisten hos hvert enkelt våpen for senere bruk.
- **CheckFireStatus**
  Holder rede på om plattformen er i en skuddsekvens. Denne funksjonen er unødvendig fordi artilleri skyter et forhåndsbestemt antall skudd med en gitt skuddrate.
- **CheckHitStatus**
  Basert på skuddraten beregnes gjennomsnittlig tid mellom hvert skudd. For å oppnå en viss spredning av granatene, er det implementert en rutine som gjør at det lander ett og bare ett skudd fra hvert våpen i løpet av denne tiden, mens det eksakte tidspunktet trekkes fra en homogen fordeling. Denne metoden finner ut om det skal ’lande’ et skudd fra et våpen i det aktuelle tidssteget, og ber så eventuelt våpenet om å vurdere effekten. Dette utføres hvert tidssteg for hvert våpen, så lenge artilleriet er aktivt.

- **Init**
  Setter startverdier for alle plattformens sentrale parametre (initierer plattformen).
- **ExecuteCommand**
  Fortsetter med gjeldende kommando til avbruddskriteriet for denne ordren er nådd.
- **GetCurrentCommand**
  Henter inn neste kommando, tolker inholdet og setter den aktiv.
- **GetDirectType**
  Returnerer verdien ”INDIRECT”.

2.8 DPlatform

DPlatform beskriver de direkteskytende plattformene. Dette er en abstrakt klasse.Alle DPlatformer har en liste med observasjonsmidler og en liste med våpen. En DPlatform kan kun skyte mot mål som er detektert av minst ett av plattformens observasjonsmidler. Hvordan observasjon og beskytning foregår, avhenger av observasjonsmiddel og våpen. To DPlatformer kan altså ha ganske forskjellige egenskaper dersom de har forskjellige våpen og forskjellige observasjonsmidler.

DPlatformer kan få flere forskjellige ordre, og behandlingen av disse ordrene foregår i klassen DPlatform, selv om ikke alle ordre kan gis til alle DPlatformer. I så tilfelle skal det komme en feilmelding som gir beskjed om at det er gitt en ”ulovlig” ordre. Behandling av ordre og plattformens status er de viktigste oppgavene til denne klassen.

2.8.1 Metoder

- **SetNewDir**
  Virtuell metode som skriver feilmelding til skjermen og avslutter programmet. Denne prosedyren skal kun kalles for de underklassene som har denne prosedyren definert lokalt (eg. TANK).
- **ReadPlatformData**
  Leser inn data fra plattformfila.
- **CheckFireStatus**
  Sjekker om plattformen har lov til å skyte. Hvis den har valgt et våpen, kaller den en prosedyre med samme navn fra dette våpenet.
- **CheckHitStatus**
  Sjekker status for avfyrte missiler fra alle plattformens våpen.
- **ExecuteCommand**
  Fortsetter med gjeldende ordre til avbruddskriterie for denne ordren er nådd.
- **GetStatus**
  Returnerer plattformens status (er den mobility-killed, firepower-killed etc.).
- **ViewUpdate**
  Forteller plattformer som har de samme to første siffer i id-nummeret hvor det er

observert fiender eller røyk. Disse plattformene kan tenkes på som organisert i samme tropp.

- **AdjustObsDevs**
  Virtuell prosedyre, som er tom. Har betydning kun for de underklassene som har denne prosedyren definert lokalt (f.eks. TANK).
- **GetCurrentCommand**
  Henter neste ordre og utfører denne.
- **SetHeight**
  Setter plattformens høyde.
- **LookAtPoint**
  Som SetNewDir.
- **IsSmoke**
  Virtuell prosedyre, som returnerer FALSE. Har betydning kun for de underklasser som har denne prosedyren som egen prosedyre.
- **TotalAmmo**
  Denne metoden returnerer den samlede mengde gjenværende ammunisjon for alle våpen på plattformen.
- **AddSmokeTarget**
  Som SetNewDir.
- **OutOfCover**
  Som SetNewDir.

2.9 TANK

Denne instanserbare klassen er ment å skulle representere stridsvogner (Strv'er) og stormpanservogner (SPV'er). Den er på mange måter lik klassen PB, som beskriver panserbekjempelsessystemer (se kapittel 2.10).

2.9.1 Metoder

- **Update**
  Kalles fra Controller hvert tidssteg og oppdaterer enheten.
- **SelectTargetAndWeapon**
  Denne prosedyren finner ut om det er mulig å beskyte noen mål, og i så fall hvilket mål som skal beskytes og hvilket våpen som skal brukes til dette. Mål som er detektert direkte, prioriteres fremfor mål der kun røyk er detektert.
- **Init**
  Initierer enheten.
- **Defend**
  Utføres dersom enheten blir beskutt, men ikke slått ut. Gir enheten økt

dekningsgrad. Her ligger også mulighet for å legge røyk, men når og hvordan røyk skal brukes, er uvisst, så denne muligheten er kommentert ut i koden.

- **SetLeaveTime**
  Fra enheten blir beskutt til den utfører metoden Defend er det en tidsforsinkelse. Denne metoden setter tidspunktet når Defend skal utføres.

- **OutOfCover**
  Utføres en viss tid etter at Defend-metoden ble utført. Denne metoden tar enheten ut av dekning igjen.

- **GetDirectType**
  Returnerer verdien ”DIRECT”.

- **AdjustObsDevs**
  Denne metoden sørger for at de observasjonsmidlene som har mulighet til det, endrer siktepunkt og siktsektor. Dette skjer dersom TANK-objektet har oppdaget en fiende eller fått beskjed av andre TANK-objekter om hvor en fiende befinner seg.

2.10 PB

Denne instanserbare klassen skal representere PB-systemer (panserbekjempelses-systemer) med tilhørende personell.

2.10.1 Metoder

- **Update**
  Kalles fra Controller hvert tidssteg og oppdaterer enheten.

- **SelectTargetAndWeapon**
  Som for samme prosedyre i TANK (se kapittel 2.9), bortsett fra at den ikke kan velge mål som er detektert vha. røyk (siden den ikke har detektert noen vha. røyk, ettersom TANK ikke avgir skuddsignatur i SIMBA).

- **Init**
  Initierer enheten.

- **Defend**
  Utføres dersom enheten blir beskutt, men ikke slått ut. PB avslutter da gjeldende ordre (som normalt er WAIT_FOR_SHOTS), og starter med neste ordre (som normalt er HIDE). Dette skal representere at en PB-stilling som blir beskutt, ikke forblir i stillingen, men prøver å komme seg vekk.

- **SetLeaveTime**
  Fra enheten blir beskutt til den utfører metoden Defend er det en tidsforsinkelse. Denne metoden setter tidspunktet når Defend skal utføres.

- **GetDirectType**
  Returnerer verdien ”DIRECT”.

2.11 ObsDev

Denne abstrakte klassen representerer det som er felles for alle observasjonsmidlene. Den inneholder blant annet targetlisten, som er en liste over alle detekterte mål. Klassen inneholder også en peker til plattformen som eier den, slik at denne ”eier-plattformen” skal være lett å ”få tak i” for observasjonsmiddelet.

2.11.1 Metoder

- **ReadFromFile**
  Leser inn data for observasjonsmiddelet fra fil (f.eks ”eye.dat”). Disse dataene angir sannsynligheten observasjonsmiddelet har for å detektere forskjellige mål. For at programmet skal fungere, må alle plattformer som et observasjonsmiddel kan detektere være definert i observasjonsmiddelets data-fil.
- **DrawTrackDetection**
  Trekker om observasjonsmiddelet mister oversikten over hvor målet er. Dette har en gitt sannsynlighet for å hende hvert tidssteg etter at målet ikke lenger kan sees.
- **GetIdentificationTimeDelay**
  Returnerer tiden mellom deteksjon og identifikasjon.
- **DrawPDetect**
  Trekker om et mål blir detektert eller ikke.
- **InTargetList**
  Tar et Platformobjekt som input. Dersom plattformen er i targetlisten, returneres det tilsvarende Targetobjektet. Ellers returneres NULL.
- **AddTarget**
  Legger til et mål i targetlisten
- **RemoveTarget**
  Fjerner et mål fra targetlisten
- **InScanList**
  Som InTargetList, men gjelder scanlist.
- **AddScan**
  Legger til et mål i scanlist.
- **RemoveScan**
  Fjerner et mål fra scanlist.
- **Init**
  Initierer plattformen.
- **SetNewDir**
  Endrer observasjonsmiddelets ViewDirection, altså retningen observasjonsmiddelet ”ser” i.

- **CheckTrackOnTarget**
  Sjekker om et gitt mål kan sees. Dersom målet er utslått, sjekkes det om dette blir oppdaget.
- **IdentificationOfTarget**
  Setter et mål til å være detektert.
- **InSector**
  Sjekker om et gitt mål er i observasjonsmiddelets ViewSector.
- **InView**
  Sjekker om det er mulig for observasjonsmiddelet å observere et bestemt mål.
- **FindSmoke**
  Observasjonsmiddelet prøver å detektere røyk fra et mål som har avgitt skuddsignatur.
- **ObservationOfTarget**
  Observasjonsmiddelet prøver å detektere alle mulige mål.
- **ChangeOldView**
  Foreløpig tom prosedyre; brukes av Eye.
- **ChangeOldSector**
  Foreløpig tom prosedyre; brukes av Eye.
- **ChangeOldElevation**
  Foreløpig tom prosedyre; brukes av Eye.

I tillegg kommer en del metoder av typen **Get*Variable***, som returnerer verdien til *Variable*, og metoder av typen **Set*Parameter***, som tilordner en verdi til *Parameter*.

2.12 Eye

Eye er en instanserbar klasse, og foreløpig den eneste underklassen til ObsDev. (Det finnes en klasse Ir også, men den er ikke i bruk.) Eye er egentlig et delvis misvisende navn på denne klassen, og bør kanskje endres. Dette observasjonsmiddelet har følgende viktige egenskaper: Det kan detektere røyk, det kan ”fokusere” på et område, og det kan midlertidig endre siktretning. Sannsynligheter for deteksjon ligger i en tekstfil for hvert enkelt Eye-objekt. Det er dette som kan være misvisende: Eye er en klasse, **ikke** et unikt objekt. Man kan altså ha flere ”Eye”-typer med forskjellige deteksjonssannsynligheter. Hvis man velger et observasjonsmiddel av typen ”EYE Ir” i SIMBA Preprosessor, er det følgelig bare datafilen som blir lest som er forskjellig fra typen ”EYE eye”. Klassen beskriver egentlig ikke et øye, men en gruppe av observasjonsmidler, hvorav øyet er ett mulig observasjonsmiddel. Eventuelle utvidelser av SIMBA vil antakelig blant annet bestå i å lage flere underklasser til klassen ObsDev.

2.12.1 Metoder

- **Init**
  Initierer den aktuelle ”Eye”-enheten.
- **Update**
  Kalles av Controller (via eierplattformen) hvert tidssteg og oppdaterer enheten.
- **SetNewDir**
  Setter ny, temporær ”ViewSector”, ”ViewDirection” og ”Elevation”. Dette betyr at observasjonsmiddelet ”fokuserer” på et område i en bestemt retning i forhold til plattformens retning.
- **LookAtPoint**
  Sørger for at ”ViewDirection” alltid peker mot observasjonsmiddelets ”ViewPoint”. Med andre ord fokuseres det her på et punkt i terenget.
- **SetOldView**
  Denne metoden kalles en bestemt tid etter SetNewDir, og setter ”ViewSector”, ”ViewDirection” og ”Elevation” tilbake til de verdiene de hadde før fokuseringen.
- **GetViewPoint**
  Returnerer verdien til Viewpoint.
- **ChangeOldView**
  Lagrer verdien av ”ViewDirection” i en backup-variabel, ”OldView”. Denne brukes senere av metoden ”SetOldView”.
- **ChangeOldSector**
  Som over, men for ”ViewSector”.
- **ChangeOldElevation**
  Som over, men for ”Elevation”.
- **AddSmokeTarget**
  Legger til et nytt mål i targetlisten ”SmokeTarget”. Dette er en targetliste for mål som kun er detektert via skuddsignatur.
- **GetSmokeTarget**
  Sjekker om det finnes mål som er detektert direkte, dvs. ikke ved hjelp av skuddsignatur.

2.13 DirWeapon

Denne klassen er en abstrakt klasse som er felles for alle direkteskytende våpen. Klassen beskriver bare våpenet, ikke plattformen.

2.13.1 Metoder

- **CalculateEffect**
  Dette er en temmelig stor og omfattende metode. Det den gjør, er å beregne

sannsynligheter for forskjellige effekter fra et skudd, og så trekke hvilken effekt som faktisk inntraff.

- **CalculatePKill**
  Beregner treff- og killsannsynligheter for et gitt skudd mot et gitt mål.
- **EqvivalentIndex**
  Finner indeks for bruk til å finne treff- og killsannsynligheter.
- **BilinearInterpol**
  Bruker bilineær interpolasjon til å finne verdien i et punkt.
- **DrawKillStatus**
  Bestemmer effekten av et skudd.
- **ReadFromFile**
  Leser data fra våpenfila.
- **AddAmmo**
  Gir våpenet ammunisjon.
- **ActiveMissiles**
  Sjekker om våpenet har et aktivt missil.
- **GetValue**
  Returnerer ”verdien” av å beskyte et mål. Denne verdien brukes av plattformen til å avgjøre hvilket mål som skal beskytes. Dette er foreløpig en veldig enkel metode, som sier at verdien av å skyte på et mål er lik sannsynligheten for at man klarer å slå det ut i løpet av en bestemt tid.

I tillegg kommer en del metoder av typen **Get*Variable***, som returnerer verdien til *Variable*, og metoder av typen **Set*Parameter***, som tilordner en verdi til *Parameter*.

2.14 Gun

Klassen Gun er en instanserbar underklasse av klassen DirWeapon. Den representerer en kanon, f.eks. på en stridsvogn. Den har blant annet de egenskaper at missilet beveger seg ”uendelig raskt”, dvs. det når målet sitt neste tidssteg, og den slipper å sikte seg inn på målet mellom hvert skudd (det må selvsagt siktes inn før første skudd).

2.14.1 Metoder

- **Init**
  Initierer enheten.
- **CheckFireStatus**
  Holder orden på hvor våpenet er i lade/sikte-prosedyren, og om våpenet er klart til å skyte.
- **CheckMissileStatus**
  Sjekker missilets effekt, tidssteget etter avfyring.

2.15 PBMissile

Denne virtuelle klassen er en underklasse av ”DirWeapon”. Den er meget liten, og kanskje et overflødig nivå. Den inneholder kun en initieringsmetode.

2.15.1 Metoder

- **Init**
  Initierer enheten.

2.16 FF

FF er en instanserbar underklasse av klassen PBMissile. Den representerer fire-and-forget PB-systemer, som for eksempel Javelin eller Gill. Felles for disse er at skytterne ikke trenger å ”følge” missilet til målet, men umiddelbart kan trekke seg ut av stillingen eller begynne å lade et nytt missil. De må sikte inn målet på nytt mellom hvert skudd - og innsiktingen kan for noen systemer ta lang tid.

2.16.1 Metoder

- **CheckFireStatus**
  Holder orden på hvor i lade/sikte-prosedyren våpenet er, og om det er klart til å skyte.
- **CheckMissileStatus**
  Beregner posisjonen til missilet, og sjekker om det har kommet frem til målet. Hvis det er tilfelle, kaller den metoden CalculateEffect.
- **CalculateEffect**
  Beregner effekten av et skudd.

2.17 Command

Denne klassen holder orden på alle kommandoene. Eller sagt på en annen måte: Hver instans av denne klassen holder orden på en kommando. Hver plattform har sin egen liste av slike instanser (de arver fra klassen Object) som samlet utgjør den fulle dreieboken for den enkelte plattformen. En siste ting det er verdt å merke seg, er at noen av parameterne i denne klassen har forskjellig betydning alt etter hvilken kommando som er lagret i den gjeldende instans.

2.17.1 Metoder

- **FileReadCommand**
  Denne metoden leser inn en kommando, og trekker ut og lagrer relevante parametre.

I tillegg kommer en mengde metoder på formen **Get*Variable***, som returnerer verdien til *Variable*.

2.18 FlagControl

Klassen FlagControl holder orden på alle flaggene, og setter deres status. Den inneholder en liste over alle Flagobjektene. Et Flagobjekt har følgende to egenskaper: Et navn og en status. Status kan settes og leses. Formålet med denne funksjonaliteten er å simulere kommunikasjon mellom plattformer.

2.18.1 Metoder

- **Start**
  Oppretter en ”Flag”-liste. Denne kalles av Controller, og bare en gang for hver kjøring av programmet (**før** loopen som kjører scenariet flere ganger).
- **Init**
  Initierer enheten (dette skjer for hver gjennomgang av scenariet, i motsetning til metoden Start).
- **AddFlag**
  Legger til et nytt Flagobjekt i listen.
- **IsSet**
  Returnerer statusen til det aktuelle Flagobjektet.
- **SetStatus**
  Setter statusen til et Flagobjekt.
- **ResetAllFlags**
  Setter statusen til alle Flagobjekter lik FALSE.

I tillegg kommer en del metoder av typen **Get*Parameter***, som returnerer verdien til *Parameter*.

2.19 Log

Alle hendelser i en simulering logges og skrives til en tekstfil. Klassen Log holder orden på denne loggingen, og er en slags liste med LogBookobjekter (se kapittel 2.20), der hvert LogBookobjekt beskriver hendelsesforløpet i én simulering.

2.19.1 Metoder

- **Init**
  Initierer enheten.
- **ReadDataFile**
  Leser fra fil hvilke typer hendelser som skal logges.
- **StartLog**
  Oppretter et nytt Logbookobjekt og legger det til i listen.
- **StopLog**
  Avslutter en logging.

- **Write*X***
  Dette er flere metoder på samme form. Alle skriver logglinjer til fil, men de skriver hver sin type logglinje. Dette er knyttet til muligheten for å la være å skrive ut enkelte typer logglinjer.

- **Add*X***
  Dette er også flere metoder på samme form. De inkrementerer alle verdien av variabelen *X*.

2.20 LogBook

Et LogBookobjekt er et element i Log, og beskriver hendelsesforløpet i én enkelt simulering.

2.20.1 Metoder

- **StartLog**
  Starter opp og initierer LogBookobjektet.

- **StopLog**
  Avslutter en logging.

- **WriteLog**
  Skriver en logglinje til fil.

- **Add*X***
  Dette er flere metoder, som alle øker verdien av variabelen *X* med 1.

2.21 Matrix

Klassen ”Matrix” representerer en matrise, og brukes av klassen Vector (se kapittel 2.26) og klassen Position (se kapittel 2.24) for å rotere vektorer, eller for å rotere posisjoner om origo.

2.21.1 Metoder

- **GetRow**
  Returnerer antall rader i matrisen.

- **GetCol**
  Returnerer antall kolonner i matrisen.

- **MatDump**
  Skriver ut matrisen til skjerm (kaller metoden MatDumpf).

- **MatDumpf**
  Skriver ut matrisen (ikke nødvendigvis til skjerm).

- **MatFillRotation**
  Lager en rotasjonsmatrise.

- **GetRotAngle**
  Returnerer rotasjonsvinkelen til en rotasjonsmatrise.

2.22 Missile

Klassen Missile er en instanserbar klasse som representerer et missil/prosjektil.

2.22.1 Metoder

- **SetLastMovingTime**
  Setter tidspunkt for missilets forrige bevegelse.
- **SetPosition**
  Setter missilets posisjon.

I tillegg kommer en del metoder av typen **Get*Parameter***, som returnerer verdien til *Parameter*.

2.23 ObList

ObList er en listestruktur som tar elementer av typen Object. De fleste klasser er derfor subklasser til klassen Object, slik at de skal kunne behandles av denne listen. ObList er en dobbeltlenket liste. Selve klassen Objekt inneholder følgelig kun pekere til forrige og neste objekt.

2.23.1 Metoder

- **AddHead**
  Legger til et objekt først i listen.
- **AddTail**
  Legger til et objekt sist i listen.
- **RemoveHead**
  Fjerner objektet som ligger først i listen.
- **RemoveTail**
  Fjerner objektet som ligger sist i listen.
- **RemoveAll**
  Fjerner alle elementene i listen.
- **Remove**
  Fjerner et gitt element i listen.
- **IsEmpty**
  Sjekker om listen er tom.
- **GetHead**
  Returnerer det første elementet i listen.

- **GetTail**
  Returnerer det siste elementet i listen.
- **GetCount**
  Returnerer antall elementer i listen.
- **InList**
  Sjekker om et gitt element finnes i listen.
- **SetCurrent**
  Dersom elementet finnes i listen, blir det satt som aktivt, og verdien TRUE returneres. Ellers returneres verdien FALSE.
- **GetCurrent**
  Returnerer aktivt element.

2.24 Position

Klassen Position representerer en posisjon i tre dimensjoner.

2.24.1 Metoder

- **Rotate**
  Roterer punktet om origo.
- **SetPos**
  Setter posisjonen
- **SetX**
  Setter X-koordinaten til posisjonen.
- **SetY**
  Setter Y-koordinaten til posisjonen.
- **SetZ**
  Setter Z-koordinaten til posisjonen.

I tillegg kommer en del metoder av typen **Get*Parameter***, som returnerer verdien til *Parameter*.

2.25 Randomgenerator

Denne klassen genererer ”tilfeldige” tall, som SIMBA blant annet bruker til å finne resultatet av skudd.

2.25.1 Metoder

- **Draw**
  Dersom denne metoden kalles uten noe argument, returnerer den et tilfeldig tall mellom 0 og 1. Dersom den kalles med et tall (mellom 0 og 1) som argument,

trekkes et tilfeldig tall mellom 0 og 1, og TRUE returneres dersom det tilfeldige tallet er mindre enn argumentet, ellers returneres FALSE.

- **DrawStdDev**

Metoden kaller Draw for å få et tilfeldig tall mellom 0 og 1. Deretter prosesseres dette tallet gjennom en formel. Sannsynlighetsfordelingen for tallet som så kommer frem er normalfordelt fra 0 til 1. Deretter ganges tallet opp med metodens eneste parameter (representerer standardavvikets størrelse) for å trekke en størrelse som er normalfordelt og ikke homogent fordelt. Formelen som benyttes er en tilnærming til den inverse kumulative normalfordelingen, og ser slik ut:

-1/2 * ln((1 - y) / y)

Denne tilnærmingen gir naturlig nok en liten feil, men den er liten i forhold til usikkerheten i de involverte størrelsene.

2.26 Vector

Denne klassen representerer en 3-dimensjonal vektor.

2.26.1 Metoder

- **SetVx, SetVy, SetVz**
  Setter henholdsvis vektorens X, Y og Z-koordinat.
- **Norm**
  Returnerer vektorens lengde.
- **Unit**
  Gjør om vektoren til en enhetsvektor som peker i samme retning som den opprinnelige vektoren.
- **Rotate**
  Roterer vektoren.
- **xyDeg**
  Returnerer vinkelen mellom øst og vektorens komponent i xy-planet
- **xyLength**
  Returnerer vektorens lengde i xy-planet.
- **SetVector**
  Setter vektoren. Kan ta forskjellige argumenter (3 koordinater, start- og sluttposisjon, eller en vinkel).

3 SIMBA PREPROSESSOR

3.1 Kort om SIMBA Preprosessor

I preprosessoren settes simuleringen opp. Det vil si at alle elementer defineres her (plattformer, våpen, observasjonsmidler, vegetasjonshekker og artilleri med tilhørende ildgivningsmekanismer (omtales i det følgende som en plattform), samt at alle plattformene instrueres med tanke på automatisert krigføring. Dette innebærer at plattformene ikke kan endre planer underveis, men i stort må følge et bestemt sett med ordre.

Dette kapitlet omhandler kildekoden til SIMBA Preprosessor og vil følgelig ikke ta for seg noen aspekter ved noen andre deler av systemet, bortsett fra der SIMBA Preprosessor deler filer med SIMBA Simulator, og filformatet derfor må samkjøres. Hensikten med dokumentet er å spesifisere implementasjonen. Dette er altså IKKE en brukerveiledning, men et hjelpedokument for dem som eventuelt skal videreutvikle SIMBA. En brukerveiledning (1) finnes i tillegg til dette dokumentet, og det anbefales på det sterkeste å lese den før man starter med den tekniske dokumentasjonen hvis man ikke allerede har kjennskap til bruken av SIMBA Preprosessor.

3.1.1 Målgruppe

Dette kapitlet er en dokumentasjon av kildekoden til SIMBA Preprosessor og ikke en opplæring i programmering, verken på generelt nivå eller i MatLab spesifikt. Det forutsettes derfor at leseren selv evner å sette seg inn i de fagfelt som skulle være påkrevd. Det forutsettes at leseren har kjennskap til programmering i MatLab, hvis ikke refereres det til MatLabs egen dokumentasjon.

3.1.2 Miljø

SIMBA Preprosessor er tilpasset MatLab 6.1 R 12.1 under MS Windows. Den skal således kunne kjøres under MatLab v.6.1 på enhver plattform. I tillegg anbefales det en stor skjerm med høy oppløsning. Det absolutte minimum er 1024x768, men 1280x1024 er det laveste som anbefales.

3.1.3 Katalogstruktur

Kildekoden til SIMBA Preprosessor må ligge i en katalog kalt *preprosessor*. Hvor denne katalogen legges er likegyldig for SIMBA Preprosessor, så det kaller vi *$PRP_ROOT*. På *$PRP_ROOT* må også katalogene *data*, *map* og *result* finnes for at alle filer skal bli plassert riktig. Mangler noen av katalogene, vil de aktuelle filene bli lagret direkte på *$PRP_ROOT*, og programmet kan i tillegg havarere når det prøver å navigere seg tilbake til *$PRP_ROOT/preprosessor* i filsystemet. Dette fordi aktiv katalog i noen tilfeller da vil være ett nivå lavere enn antatt. De nødvendige katalogene brukes slik:

data: Her lagres kommandofiler, elementfiler og scenariofiler
map: Kartfiler og eventuell wallfil lagres her
result: Katalogen for simuleringsresultater opprettes her

Legg spesielt merke til at katalogen *result* ikke egentlig er nødvendig for SIMBA Preprosessor sin del. Det er litt bakvendt at katalogen for resultater fra SIMBA Simulator skal spesifiseres i SIMBA Preprosessor, men slik er det. Og når det først er sånn, så er selvsagt koden skrevet slik at programmet får hikke hvis denne katalogen ikke eksisterer. Nå finnes det jo selvsagt også en god grunn for hvorfor denne katalogen spesifiseres der den gjør. Og det er at informasjonen skal

lagres i scenariofilen. SIMBA Postprosessor og SIMBA Kartlogg er nemlig også avhengig av denne katalogen og de data som SIMBA Simulator legger der. Men når det er sagt, så understrekes det altså at katalogen ikke er påkrevd for å kun kjøre SIMBA Preprosessor.

3.2 Resultatfilsett

Resultatet av arbeidet som utføres i SIMBA Preprosessor lagres på et antall filer. Disse filene hentes så inn i SIMBA Simulator for simulering av det oppsatte scenariet. SIMBA Preprosessor returnerer alltid minst tre filer. Hvis instruert om det, kan den returnere fem. Samtlige lagres i ASCII format og er fullt leselige for mennesker hvis man bare vet hva som står der. De fire filene som kan produseres ved hjelp av SIMBA Preprosessor er:

- ✓ Scenariofil (*.scn): beskriver hvordan verden ser ut (kartfiler o.l.)
- ✓ Elementfil (*.el): beskriver de stridende parter (plattformer, våpen, o.s.v.)
- ✓ Kommandofil (*.cm): beskriver elementenes oppførsel (dreiebok)
- ✓ Wallfil (*.wall): valgfri fil for definisjon av kunstige siktbegrensninger
- ✓ Logsetupfil (*.log): spesifiserer hva som skal skrives til logfilene

SIMBA Preprosessor vil lagre de fire første av disse filene med samme fornavn, slik at man lett kan se hvilke filer som hører sammen. SIMBA Simulator krever også at dette er tilfellet når en simulering skal startes, ettersom det kun er navnet på scenariofilen som må oppgis av bruker. Filnavnendelsene er standardiserte, og de angitte endelsene er de som forlanges av SIMBA Simulator. Logsetupfilen kan ha et hvilket som helst fornavn, og dette lagres i scenariofilen. Dette er gjort fordi man typisk vil bruke samme fil i mange scenarier.

3.3 Kartfilsett

Det som omtales som ”kartet” i sammenheng med SIMBA Preprosessor, består av 4 filer. De sier hver på sin måte noe om det terrenget simuleringen skal foregå i. Med unntak av kartbildefilen brukes samtlige også av SIMBA Simulator. I tillegg finnes det en matrisefil som regnes ut på grunnlag av data i de andre filene.

3.3.1 Kartbildefil

Kartbildefilen skal ha filnavnendelsen *jpg*. Den inneholder et bilde av et kart med sjøer, fjell og gjerne noen veier. Poenget er at det blir noe enklere å orientere seg med et kartbokblad enn med et høydekoteplott. Hvordan dette bildet genereres, er for så vidt likegyldig for SIMBA Preprosessor, men det som i all hovedsak har vært brukt til nå, er rett og slett veikart som digitaliseres. Uansett hvor det kommer fra, så må kartet, som filnavnendelsen antyder, lagres på det standardiserte JPEG-formatet. Denne filen er ikke påkrevd for å kjøre SIMBA Preprosessor. Visning av digitalt kart spesifiseres via programmets meny.

3.3.2 Terrengfil

Det tidligere omtalte høydekoteplottet genereres på bakgrunn av innholdet i denne filen, som skal ha filnavnendelsen *ter*. Avstanden mellom punktene i terrengfilen angis i Kartdatafilen. Informasjonen lagres digitalt. Denne filen kan altså ikke uten videre leses av oss mennesker. I tillegg bør det legges merke til at høydepunktene strekkes for å tilpasses størrelsen på vinduet. Dette kan medføre noe avvik i forhold til informasjonen i kartbildefilen. I praksis betyr det at en kilometer i nord/syd retning ikke er like lang på skjermen som en kilometer i øst/vest retning. Det kan se litt rart ut på skjermen, men det regnes riktig likevel.

3.3.3 Vegetasjonsfil

Denne filen skal ha filnavnendelsen *veg*, og må ikke forveksles med Wallfilen. Spesielt kan en slik misforståelse oppstå fordi konstruksjon av Wallfiler gjøres via menyvalget *Vegetation/Make* i SIMBA Preprosessor sitt Hovedvindu. Filen skal inneholde data for vegetasjonens høyde i det aktuelle området, og skal være bygget opp på samme måte som Terrengfilen med samme avstand mellom punktene, men med ett punkt mindre i hver retning. Her er det nemlig vegetasjonens høyde i senter av et rektangel dannet av fire punkter i terrengfilen som skal spesifiseres. For øvrig gjelder de samme bemerkningene angående format, tilnærminger og avvik. Vegetasjonsfilen er ikke påkrevd, og ettersom de nødvendige data har vært vanskelig å få tak i, har denne funksjonaliteten vært lite brukt.

3.3.4 Kartdatafil

Innholdet i denne filen angir hvor i verden kartet er fra, hvor stort område det dekker, og hvilken oppløsning punktene i Terrengfilen og Vegetasjonsfilen har. Filen skal ha etternavn *dat*, og er den eneste av filene i Kartfilsettet som skrives (og leses) på ASCII-format.

3.3.5 Matrisefil

Denne filen fylles med data regnet ut på bakgrunn av Terrengfilen. Dette gjøres for å slippe å vente på utregning av nødvendige størrelser hver gang man laster et Kartfilsett. Finnes denne filen, brukes den direkte. Filen skal ha etternavn *con*, og den opprettes automatisk. En smule varsomhet må utvises ved endring av et eksisterende kart, eventuelt opprettelse av et nytt kart med samme navn som et eksisterende, fordi det ikke foretas noe kontroll før Matrisefilen aksepteres og benyttes. Man kan altså ende opp med å bruke Matrisefilen fra et gammelt kart i slike tilfeller.

3.4 Plattformer og elementer

Alle enheter som inngår i en simulering kalles med en felles betegnelse for elementer. Disse enhetene omfatter altså både plattformer, våpen og observasjonsmidler. Siden dette er en regel og det ikke finnes regler uten unntak, så er de kunstige siktbegrensningene som lagres i en Wallfil ikke å anse som elementer. Plattformer er definert som baser for ildgivning og omfatter også indirekteskytende baser som artilleri. Artilleri er forøvrig den eneste indirekte plattformtypen for øyeblikket. Man ser for seg mulige utvidelser av SIMBA til også å dekke miner og fjernstyrte våpen. Utskytningsrampen for disse (eller personen som holder våpenet, eller minfeltet som helhet) vil i så fall også være å anse som plattformer med den terminologien som er brukt i SIMBA Preprosessor og i dette dokumentet. Våpen er definert som selve ildgivningsinstrumentet; kanonen eller geværet om man vil.

Det som på en eller annen måte står for måldeteksjon defineres som et observasjonsmiddel. Foreløpig finnes kun normale og infrarøde observasjonsmidler, men sammen med definisjonen av plattformer vil definisjonen av observasjonsmidler bety at for eksempel en observasjonspost for fjernstyring av indirekte våpen blir å anse som et observasjonsmiddel, og ikke som en plattform, fordi den er en del av ildgivningsmekanismen.

3.4.1 Plattformer

De plattformer som støttes er TANK, PB og ARTILLERY. Alle anses å ha en posisjon, men hva som defineres som posisjon varierer noe. I tillegg er det noen forskjeller i sårbarheten.

3.4.1.1 TANK-plattformer

Denne plattformkategorien betegner pansrede kjøretøy. Alt fra stormpanservogner til mineryddingsfartøyer og tunge stridsvogner vil komme inn her. Men legg merke til at det kun er

basen for våpenet som omfattes. Våpenet selv (eller plogen eller hva det måtte være) defineres som et våpen som legges til plattformen. Det bemerkes for ordens skyld at miner og minerydding ikke er implementert.

3.4.1.2 PB-plattformer

Betegnelsen for personbaserte panserbekjempelsesstillinger er PB. Det kan dreie seg om ett eller flere mennesker som kan betjene ett eller flere våpen (våpnene angis separat). Ettersom SIMBA ikke opererer på personnivå, er antallet personer ikke videre interessant. Med de våpen som antas brukt slår man ut hele PB-stillinger eller så forblir de uskadet. Det gås ikke inn på hvor mange menn (evt. kvinner) som ble tatt ut, for så å vurdere om et våpen kan betjenes eller ikke.

3.4.1.3 ARTILLERY-plattformer

Dette er betegnelsen for artilleri. I våre studier har vi hittil ikke vært interessert i å se på muligheten av å slå ut artilleristillinger. Vi har bare inkludert artilleri for å se på synergieffekter. Følgelig har vi ikke hatt behov for å definere sårbarhet for artilleriet, og dermed har ikke den egentlige posisjonen til kanonene vært av særlig interesse. Derfor har artilleriets siktepunkt blitt å anse som artilleriets posisjon. Ingen våpen kan skyte på et artilleri, og ingen observasjonsmidler kan oppdage det.

3.4.2 Våpen

Definert som ildgivningsmekanismer omfatter dette alt fra geværer og artilleriløp til miner og raketter; kort sagt alt som kan gjøre skade på mennesker og materiell. Legg merke til at SIMBA ikke simulerer på personnivå, samt at minefunksjonalitet ikke er implementert. Hvert våpen har sin datafil som angir sannsynlighet for skade mot alle plattformer vi ser for oss at det aktuelle våpen skal skyte mot. Disse filene benyttes ikke i SIMBA Preprosessor, men de må finnes for at SIMBA Simulator skal ha mulighet til å opptre rasjonelt. I så måte er det viktig at det ikke settes opp elementer i SIMBA Preprosessor som ikke støttes i disse filene. Dette begrenses ved at alle elementer velges fra lister, i motsetning til å skrives inn manuelt.

3.4.3 Observasjonsmidler

Her er det snakk om systemer som kan brukes til deteksjon av fiender. Alt fra radar til satellittovervåkning vil komme inn her. Det eneste som er implementert er imidlertid vanlig sikt (øyne) og en infrarød variant av dette. Hvert observasjonsmiddel har sin datafil som angir sannsynligheten for deteksjon av alle plattformer de antas å kunne komme til å se. Disse filene benyttes ikke i SIMBA Preprosessor, men de må finnes for at SIMBA Simulator skal ha mulighet til å opptre rasjonelt. I så måte er det viktig at det ikke settes opp elementer i SIMBA Preprosessor som ikke støttes i disse filene. Dette begrenses ved at alle elementer velges fra lister, i motsetning til å skrives inn manuelt.

3.5 Vinduer og dialogbokser

All interaksjon mellom SIMBA Preprosessor og brukeren forgår via spesifikke vinduer og dialogbokser. Det vil si at det skal være unødvendig å benytte kommandolinje eller tilleggsprogrammer. Alt som presenteres av data, hovedsakelig data som brukeren allerede har tastet inn eller hentet fra fil, vises i dertil egnede vinduer. All informasjon som brukeren må eller rasjonelt sett kan ha lyst til å mate inn i SIMBA Preprosessor, skal hentes ved hjelp av dialogbokser. Det finnes stort sett en dialogboks for hver type data, og mange benytter i tillegg globale variable for å overføre data.

Det er ikke er benyttet funksjoner av typen *onCreate ()*, eller *CreateFcn ()* som de heter i MatLab, det vil si funksjoner som automatisk kjøres når et vindu eller en dialogboks opprettes, men før den vises på skjermen. Dette medfører noen forfunksjoner uten tilknytning til selve

vinduet. Som eksempel kan vi nevne Plattformdialogboksen. Åpnes dette vinduet fra MatLabs kommandoprompt, vil man se en dialogboks med en editboks og to tomme listebokser (samt to knapper og fire tekstbokser, men de er ikke interessante i denne sammenheng). I koden vil denne dialogboksen alltid åpnes via funksjonen *showaddplatforms ()*, som har det tunge ansvaret å putte data i listeboksene. I dokumentasjonen under er det forsøkt angitt slike forfunksjoner. For øvrig later det til at en figur er en figur i MatLabs øyne. Ettersom en utvikler er mer interessert i funksjonalitet enn i form, skilles det likevel mellom vinduer og dialogbokser i denne dokumentasjonen.

3.5.1 Vinduer

Fra det ovenstående har vi altså at vinduer brukes til mer eller mindre permanent visning av data. Det er ingen spesiell grunn til å gjøre disse modale, med mindre de også i noen grad brukes til innhenting av data fra bruker (for eksempel et klikk på et kart), eller at det foregår beregninger som forventes å strekke seg så pass frem i tid at det kan tenkes at brukeren kan bli utålmodig og vil se på noe annet imens.

3.5.1.1 Hovedvinduet

Vinduet der kartet skal vises kalles hovedvinduet, og det er definert i filen *sp.m*. Programmet startes ved å aktivisere denne filen. Det er også i hovedvinduet menyen ligger (øverst oppe under tittellinjen). Umiddelbart etter oppstart vises det et bilde av en liten løve i kartområdet. Under kartet (evt. løveungen) finner vi meldingsboksen som er en standard tekstboks (noe mørkere farge enn resten av bakgrunnen). Her vises diverse meldinger om hva som forventes av brukere, eventuelt indikasjoner på hvor langt SIMBA Preprosessor har kommet med det den holder på med for øyeblikket. Legg merke til at det ikke finnes noen close-knapp eller tilsvarende menyvalg.

Hovedmenyen
Hovedmenyen er delt opp i 4 kategorier. Implementasjonen ligger i filen som definerer vinduet (*sp.m*). Menyen er strukturert som følger:

1) Map
Her finner du tre valg som alle har med kartfremvisning å gjøre:

a) **Load**
Callback: *global mainhandle, mainhandle = gcbf; loadmap*
Kaller *loadmap ()* (3.5.2.11) etter å ha satt den globale variabelen *mainhandle* (3.6.10) lik handle til hovedvinduet.

b) **Show**
Valget åpner en ny meny som inneholder opsjoner for å styre hvordan det aktive kartet skal vises. Følgende valg finnes:

i) **Contourplot**
En ny rullegardin kommer frem. Denne med kun to valg:

(1) **without veg**.
Callback: *showmap con*
Kaller *showmap ()* (3.8.48) med *con* som argument. Resultatet blir identisk med notasjonen *showmap ('con')*.

(2) **with veg**.
Callback: *showmap conveg*
Kaller *showmap ()* (3.8.48) med *conveg* som argument. Resultatet blir identisk med notasjonen *showmap ('conveg')*.

ii) **Digital map**
Callback: *showmap scanned*
Kaller *showmap ()* (3.8.48) med *scanned* som argument. Resultatet blir identisk med notasjonen *showmap ('scanned')*.

iii) **Pcolor**
Callback: *showmap p*
Kaller *showmap ()* (3.8.48) med *p* som argument. Resultatet blir identisk med notasjonen *showmap ('p')*.

iv) **Surf**
Callback: *showmap su*
Kaller *showmap ()* (3.8.48) med *su* som argument. Resultatet blir identisk med notasjonen *showmap ('su')*.

v) **SurfL**
Callback: *showmap sul*
Kaller *showmap ()* (3.8.48) med *sul* som argument. Resultatet blir identisk med notasjonen *showmap ('sul')*.

vi) **Mesh**
Callback: *showmap me*
Kaller *showmap ()* (3.8.48) med *me* som argument. Resultatet blir identisk med notasjonen *showmap ('me')*.

c) **Zoom**
Zoom-menyen inneholder valg for å forstørre kartutsnitt. Der finnes tre valg, men det gjøres oppmerksom på at det er noe usikkert om denne funksjonaliteten virker på en tilfredsstillende måte, da den har vært lite utprøvd.

i) Zoom out
Callback: *CB zoomout*
Kaller *CB ()* (3.8.6) med *zoomout* som argument. Resultatet blir identisk med notasjonen *CB ('zoomout')*.

ii) Zoom in
Callback: *CB zoomin*
Kaller *CB ()* (3.8.6) med *zoomin* som argument. Resultatet blir identisk med notasjonen *CB ('zoomin')*.

iii) Show all
Callback: *CB all*
Kaller *CB ()* (3.8.6) med *all* som argument. Resultatet blir identisk med notasjonen *CB ('all')*.

2) Platforms
Alle menyvalg som omhandler elementene i simuleringen skal være samlet her. Det finnes bare to valg på denne menyen:

a) Load
Callback: *global mainhandle, mainhandle = gcbf; loadel*
Kaller *loadel ()* (3.8.25) etter å ha satt den globale variabelen *mainhandle* (3.6.10) lik handle til hovedvinduet.

b) Control
Callback: *global mainhandle, mainhandle = gcbf; showelements*
Kaller *showelements ()* (3.8.47) etter å ha satt den globale variabelen *mainhandle* (3.6.10) lik handle til hovedvinduet. *showelements ()* klargjør og åpner Element Control (3.5.1.2).

3) Visibility
Visibilitymenyen har tre valg som alle handler om sikt.

a) Profile plot
Callback : *viz profile*
Kaller *viz ()* (3.8.54) med *profile* som argument. Resultatet blir identisk med notasjonen *viz ('profile')*.

b) In sector
Callback: *global mainhandle, mainhandle = gcbf; viz sector*

Kaller *viz ()* (3.8.54) med *all* som argument, etter å ha satt den globale variabelen *mainhandle* (3.6.10) lik handle til hovedvinduet. Resultatet blir identisk med notasjonen *viz ('all')*.

c) Along line
Callback : *viz line*
Kaller *viz ()* (3.8.54) med *line* som argument. Resultatet blir identisk med notasjonen *viz ('line')*.

4) Vegetation
Denne menyen har bare ett valg. Og det gjelder ikke vegetasjon, men hekker (walls) som skal gi den effekten på sikt som vegetasjonen egentlig har, bare for å ha nevnt det.

a) Make
Callback: *get_veg_height*
Kaller *get_veg_height ()*. Dette er en dialogboks, og funksjonen som ender opp med å gjøre arbeidet er *makeveg ()* (3.8.29).

Kartområdet
Kartområdet er det området der enten kartet eller bildet av ungløven Simba vises. Når løven er fremme, har dette området et callback: *global mainhandle, mainhandle = gcbf, loadmap*. Med andre ord: Den globale variabelen *mainhandle* settes lik hovedvinduets handle før *loadmap ()* kalles. Når det er et kart i dette området, styres input av den funksjon som til enhver tid måtte trenge data fra kartet. Det dreier seg stort sett om innhenting av punkter. Området er definert som et aksesystem.

Meldingsområdet

Som nevnt tidligere, er dette det mørke feltet nederst i vinduet, under Kartområdet. Dette er definert som en tekstboks, og tekstbokser har ingen callback. Verdien som vises på skjermen er objektets String parameter, som kan settes fra hvor som helst. Skulle man ha behov for å finne ut hvor en melding kommer fra, anbefales tekstsøk i samtlige ".m" filer.

3.5.1.2 Element Control

Dette vinduet er definert i filen *ec.m* og består av tre signifikante områder. Øverst har vi Elementområdet bestående av tre tekstbokser, tre listebokser og tre ganger tre knapper merket Add, Mod og Del. Så kommer Kommandoområdet som består av de tre rullegardinboksene og tekstboksene umiddelbart over disse, samt den store listeboksen ute til høyre med tilhørende tekstboks og to knapper merket Delete og Delete all. Det siste området er Handlingsområdet. Handlingsområdet utgjør den siste delen av vinduet; Tre pluss to knapper, samt et aksekors til fremvisning av plattformsymboler. Inndelingen i tre områder er ikke på noen måte representert i kildekoden. De benyttes her for å lette arbeidet med å beskrive elementene i dette dokumentet.

I Elementområdet finner vi Plattformlisten, Våpenlisten og Observasjonsmiddellisten. Hver av disse har tre knapper under listen. Disse refereres til ved hjelp av teksten på knappene sammen med listenavnet. Det samme gjelder tekstboksen i overkant, i den grad de omtales. Formatet blir da for eksempel Plattformlistens addknapp eller våpenlistens tekstboks.

I Kommandoområdet har vi som nevnt tre rullegardiner som benevnes etter tittelen på tekstboksen over, samt kommandolisten med tilhørende sletteknapper og tekstboks. Referansene bygges opp som for listene i Elementområdet.

Det siste området er Handlingsområdet. Knappene refereres til ved hjelp av teksten som finnes på dem, og aksekorset kalles bare…. "Aksekorset".

Plattformlistens Tekstboks
Tag: StaticText2
Callback: Ingen
Handling: Dette er kun en ledetekst for Plattformlisten.
Plassering: Øverst til venstre i Elementområdet, over Plattformlisten. Har teksten 'Current platforms :'.

Plattformlisten
Tag: platformlist
Callback: *changeplatform*
Handling: Kaller *changeplatform ()* (3.8.6), som oppdaterer resten av vinduet i tråd med endringen i indeksert element i Plattformlisten.
Plassering: Til venstre i Elementområdet under Plattformlistens Tekstboks.

Plattformlistens Addknapp
Tag: AddPlatform
Callback: *global elhandle; elhandle = gcbf; showaddplatforms*
Handling: Lagrer handle til Element Control i den globale variabelen *elhandle*. Dette er sannsynligvis unødvendig ettersom denne verdien skal være lagret i den globale variabelen *cmdhandle* (3.6.1) allerede. Deretter kalles *showaddplatforms ()* (3.8.45), som klargjør og starter dialogboksen for tillegging av plattformer.
Plassering: Den venstre av de tre knappene under Plattformlisten.

Plattformlistens Modknapp
Tag: ModPlatform
Callback: *showmodplatform*
Handling: Kaller *showmodplatform ()* (3.8.50) som klargjør og åpner dialogboksen for endring av data for allerede registrerte plattformer.
Plassering: Den midterste av knappene umiddelbart under Plattformlisten

Plattformlistens Delknapp
Tag: DelPlatform
Callback: *DeleteElement (1)*
Handling: Kaller *DeleteElement ()* (3.8.16) som er en samlefunksjon for sletting av elementer. Tallet sier fra til funksjonen at kallet kommer fra knappen relatert til plattformer.
Plassering: Knappen lengst til høyre av de tre knappene umiddelbart under Plattformlisten

Våpenlistens Tekstboks
Tag: StaticText2
Callback: Ingen
Handling: Dette er kun en ledetekst til Våpenlisten
Plassering: Tekstboksen er plassert like over den midterste av listeboksene i Elementområdet. Inneholder teksten *Weapons on platform :*.

Våpenlisten
Tag: weaponlist
Callback: Ingen
Handling: Viser hvilke våpen som er tilordnet den plattformen som er indeksert i Plattformlisten. Indeksen i Våpenlisten avgjør hvilket våpen eventuelle kommandoer fra Våpenkommandolisten refererer til, samt hvilket våpen som skal endres eller slettes hvis brukeren gjør slike valg.

Plassering: Den midterste listeboksen i Elementområdet

Våpenlistens Addknapp
Tag: AddWeapon
Callback: *global elhandle, elhandle = gcbf; showaddweapons*
Handling: Lagrer handle til Element Control vinduet (3.5.1.2) i den globale variabelen *elhandle*. Dette er sannsynligvis unødvendig ettersom denne verdien skal være lagret i den globale variabelen *cmdhandle* (3.6.1) allerede. Deretter kalles *showaddweapons ()* (3.8.46) som klargjør og starter dialogboksen for valg av våpen som skal legges til den aktive plattformen.
Plassering: Den venstre av de tre knappene under Våpenlisten.

Våpenlistens Modknapp
Tag: ModWeapon
Callback: *showmodweapon*
Handling: Kaller *showmodweapon ()* (3.8.51) som klargjør og åpner dialogboksen for endring av allerede spesifiserte våpen.
Plassering: Den midterste av de tre knappene umiddelbart under våpenlisten

Våpenlistens Delknapp
Tag: DelWeapon
Callback: *DeleteElement (2)*
Handling: Kaller *DeleteElement ()* (3.8.16), som er en samlefunksjon for sletting av elementer. Tallet sier fra til funksjonen at kallet kommer fra knappen relatert til våpen.
Plassering: Knappen lengst til høyre av de tre umiddelbart under Våpenlisten.

Observasjonsmiddellistens tekstboks
Tag: StaticText2
Callback: Ingen
Handling: Dette er kun en ledetekst for observasjonsmiddellisten.
Plassering: Like over Observasjonsmiddellisten. Teksten i boksen er "*Obs.devs. on Platform :*".

Observasjonsmiddellisten
Tag: obsdevlist
Callback: Ingen
Handling: Viser hvilke observasjonsmidler som er tilordnet den plattformen som er indeksert i Plattformlisten. Indeksen i Observasjonslisten avgjør hvilket observasjonsmiddel eventuelle kommandoer fra Observasjonsmiddelkommandolisten refererer til, samt hvilket observasjonsmiddel som skal endres eller slettes hvis brukeren gjør slike valg.
Plassering: Listeboksen lengst til høyre i Elementområdet

Observasjonsmiddellistens Addknapp
Tag: AddObsdev
Callback: *global elhandle, elhandle = gcbf; showaddobsdevs*
Handling: Lagrer handle til Element Control i den globale variabelen *elhandle*. Dette er sannsynligvis unødvendig, ettersom denne verdien skal være lagret i den globale variabelen *cmdhandle* (3.6.1) allerede. Deretter kalles *showaddobsdevs ()* (3.8.44), som klargjør og starter dialogboksen for valg av observasjonsmidler som skal legges til den aktive plattformen.
Plassering: Den venstre av de tre knappene under Observasjonsmiddellisten.

Observasjonsmiddellistens Modknapp
Tag: ModObsdev
Callback: *showmodobsdev*
Handling: Kaller *showmodobsdev ()* (3.8.49) som klargjør og åpner dialogboksen for endring av allerede spesifiserte observasjonsmidler.
Plassering: Den midterste av de tre knappene umiddelbart under Observasjonsmiddellisten.

Observasjonsmiddellistens Delknapp
Tag: DelObsdev
Callback: *DeleteElement (3)*
Handling: Kaller *DeleteElement ()* (3.8.16), som er en samlefunksjon for sletting av elementer. Tallet sier fra til funksjonen at kallet kommer fra knappen relatert til observasjonsmidler.
Plassering: Knappen lengst til høyre av de tre umiddelbart under Observasjonsmiddellisten.

Plattformkommandolistens tekstboks
Tag: StaticText1
Callback: Ingen
Handling: Dette er kun en statisk ledetekst: ”Plattform :”
Plassering: Helt til venstre under Plattformlistens Addknapp.

Plattformkommandolisten
Tag: platformcommandpopup
Callback: *elcommand (1, get (gco, ’Value’))*
Handling: Når brukeren gjør et valg i pop-up menyen, vil *elcommand ()* (3.8.20) kalles med to parametre. Det første er et fast tall som spesifiserer avsender, og det andre er nummeret på det valgte elementet i pop-up menyen.
Plassering: Umiddelbart under Plattformkommandolistens tekstboks. Den venstre av pop-up menyene (eller drop-down menyene om man vil).

Våpenkommandolistens tekstboks
Tag: StaticText1
Callback: Ingen
Handling: Dette er kun en statisk ledetekst: "Weapon :"
Plassering: Til høyre for Plattformkommandolistens tekstboks

Våpenkommandolisten
Tag: weaponcommandpopup
Callback: *elcommand (2, get (gco, ’Value’))*
Handling: Når brukeren gjør et valg i pop-up menyen, vil *elcommand ()* (3.8.20) kalles med to parametre. Det første er et fast tall som spesifiserer avsender, og det andre er nummeret på det valgte elementet i pop-up menyen.
Plassering: Umiddelbart under Våpenkommandolistens tekstboks. Den midterste av pop-up menyene (eller drop-down menyene om man vil).

Observasjonsmiddelkommandolistens tekstboks
Tag: StaticText1
Callback: Ingen
Handling: Dette er kun en statisk ledetekst: ”Obs.dev. :”
Plassering: Til høyre for Våpenkommandolistens tekstboks

Observasjonsmiddelkommandolisten
Tag: obsdevcommandpopup
Callback: *elcommand (2, get (gco, ’Value’))*

Handling: Når brukeren gjør et valg i pop-up menyen, vil *elcommand ()* (3.8.20) kalles med to parametre. Det første er et fast tall som spesifiserer avsender, og det andre er nummeret på det valgte elementet i pop-up menyen.
Plassering: Umiddelbart under Observasjonsmiddelkommandolistens tekstboks. Den høyre av pop-up menyene (eller drop-down menyene om man vil).

Kommandolistens tekstboks
Tag: StaticText3
Callback: Ingen
Handling: Dette er kun en statisk ledetekst: ”Command list :”.
Plassering: Ytterst til høyre, på høyde med pop-up menyenes listebokser.

Kommandolisten
Tag: commandlist
Callback: Ingen
Handling: Denne listeboksen brukes kun til å vise frem hvilke kommandoer som er gitt til en enkelt plattform. Det kan imidlertid markeres i listen, og det er det markerte elementet som styrer hva som skal slettes ved hjelp av Kommandolistens Deleteknapp, samt hvor eventuelle nye kommandoer skal settes inn i listen (settes inn etter den markerte kommandoen).
Plassering: Til høyre for pop-up menyene. Helt ute ved kanten av vinduet.

Kommandolistens Deleteknapp
Tag: deletecommand
Callback: *elcommand (’delete’)*
Handling: Sletter ugjenkallelig den markerte kommandoen fra Kommandolisten. For direkteskytende plattformer tillates den første kommandoen (START-linjen) kun slettet hvis det er den eneste kommandoen i Kommandolisten. Forsøkes denne slettet med flere kommandoer i Kommandolisten, eller knappen aktiveres uten kommandoer i Kommandolisten, skjer det overhode ingenting.
Plassering: Under Kommandolisten. Den høyre av de to knappene.

Kommandolistens Deleteallknapp
Tag: deleteallcommands
Callback: *elcommand (’deleteall’)*
Handling: Tømmer kommandolisten. Her foregår det ingen tolkning for å kontrollere om dette er lurt eller nødvendig. Listen simpelthen bare tømmes.
Plassering: Under Kommandolisten. Den venstre av de to knappene.

Copyknappen
Tag: copybutton
Callback: *copycurrent ()*
Handling: Aktivering av denne knappen medfører en kloning av den plattformen som måtte være merket/valgt/aktivert i Plattformlisten. Den nye plattformen, som altså blir nøyaktig lik den aktive, inklusive kommandoliste og plattform ID’en, legges til nederst i Plattformlisten. Denne knappen er tenkt brukt ved produksjon av stridsvognskolonner, der flere stridsvogner skal gis lik oppførsel.
Plassering: Øverst til venstre i Handlingsdelen av vinduet. Altså under Plattformkommandolisten og helt til venstre.

Cleanupknappen
Tag: Pushbutton2
Callback: *cleanup ()*

Handling: Denne knappen brukes for å få bedre oversikt over hvor langt man har kommet i prosessen med å sette opp et scenario. Alle plattformer tegnes opp på nytt, men kun i siste kjente posisjon.
Plassering: Rett under Copycurrentknappen.

Redrawknappen
Tag: Pushbutton6
Callback: *global plottype mainhandle*
message('Redrawing, please wait ...')
figure(mainhandle);
showmap(plottype);
elplot2
Handling: Som man ser av callbacket, sendes det en melding til brukeren (via Hovedvinduet) før kartet tegnes på nytt og alle elementene tegnes oppå der igjen. Denne knappen brukes altså til å få tegnet opp kartet med plattformer på nytt, ut i fra hvilke kartvalg og hvilke kommandoer og plattformer som finnes for øyeblikket.
Plassering: Til høyre for Cleanupknappen og under Aksekorset.

Aksekorset
Tag: Axes1
Callback: Ingen
Handling: Her vises et symbol som representerer typen til den plattformen som er markert i Plattformlisten. For tiden er alle plattformtyper representert ved det samme symbolet (unntak: Artilleri); en fylt sirkel. Fargen som sirkelen fylles med avhenger av (eller representerer) hvilken side plattformen er på; venn eller fiende. Venn representeres med blå farge og fiende med oransje. Legg også merke til at de samme symbolene brukes på kartet i Hovedvinduet (3.5.1.1). Artilleri vises med plattformnummer og en prikk på siktepunktet. Fargen på plattformnummeret representerer da sidetilhørighet.
Plassering: Til høyre for Copyknappen og over Redrawknappen.

Savefilesknappen
Tag: Pushbutton5
Callback: *Savefile ()*
Handling: Initierer lagring av Resultatfilsettet. Hvis alt går etter planen, vil filene inneholde en representasjon av det arbeidet som er gjort i SIMBA Preprosessor. SIMBA Preprosessor vil utbe seg en del katalogspesifikk informasjon før filene faktisk blir lagret, men denne knappen er eneste måten å starte lagringsprosessen på fra brukergrensesnittet.
Plassering: Nederst til venstre.

Closeknappen
Tag: Pushbutton1
Callback: *delete (gcbf)*
Handling: Lukker vinduet uten å lagre til fil. Alle data og alt arbeid som er utført vil imidlertid være lagret i variabelen *elements* ettersom denne er global. Åpnes vinduet på nytt, vil innholdet i *elements* brukes til å initialisere med. Følgelig må man slette *elements* manuelt eller stenge ned MatLab (og starte den igjen) hvis det man ønsker er å starte med blanke ark. Innlasting av scenario fra fil vil selvsagt overskrive gamle data i *elements*.
Plassering: Til høyre for Savefilesknappen

3.5.2 Dialogbokser

Etter definisjonen i toppen av dette kapittelet er alle vinduer som bare henter data fra brukeren dialogbokser. Disse finnes det svært mange av i SIMBA Preprosessor. Slik koden ligger, er det mer eller mindre en dialogboks for hver data-entitet som SIMBA Preprosessor kan få bruk for. Det typiske er at de også har hver sin globale variable til å putte returdata i. Mange har enda til flere slike globale returvariable. Slike variable er nevnt for å lette arbeidet med å spore dataflyten i SIMBA Preprosessor. Videre er mange elementer i dialogvinduene beskrevet etter modellen fra kapittelet som beskriver vinduene (over). Dette hovedsakelig for å gi oversikt over hvilke callbacker som aktiveres av hvilke grensesnittelementer. Husk at alle vinduer, og dermed også dialogbokser, er definert ved hjelp av 2 filer, en **.m* fil og en **.fig* fil.

3.5.2.1 Plattformdialogboksen

Filnavn: AddElement
Tag: AddElem
Returvariable: Bruker den globale strukturen *elements* (3.6.3)
Bruksområde: Denne dialogboksen brukes både ved innlegging av nye plattformer og ved endring av eksisterende plattformer. Den kalles alltid via en såkalt forfunksjon, og i dennes tilfelle er det to forskjellige som benyttes. Den første funksjonen er *showaddplatforms ()* (3.8.45). Denne funksjonen åpner dialogboksen og fyller inn data fra filer. Den andre funksjonen er *showmodplatform ()* (3.8.50), og den har noe mer å gjøre. I tillegg til å laste inn de samme data fra fil, endrer den ledetekstene og vindustittelen slik at de blir tilpasset bruksområdet som er å modifisere plattformer, samt at den endrer callbacket på OK-knappen.

Tittellinjen
Tag: Aksesseres som parameter til vinduet.
Callback: Ingen
Handling: Teksten skifter etter hvilken sammenheng vinduet åpnes i. Når kallet kommer fra *showaddplatforms ()* (3.8.45) og hvis dialogboksen kalles direkte, er tittelen på vinduet 'New element'. Når det åpnes fra *showmodplatform ()* (3.8.50), endres tittelen til 'Modify element'. Utover dette har ikke tittellinjen noen aktiv rolle
Plassering: På toppen av vinduet.

Brukerinformasjon
Tag: StaticText1
Callback: Ingen
Handling: Forteller brukeren hva som forventes. For øvrig et helt bortkastet felt, ettersom tittellinjen kunne vært brukt til dette formålet, samt at brukeren som regel vet hvorfor dialogboksen ble åpnet i utgangspunktet. Teksten skal være henholdsvis 'Add Element' eller 'Modify element'.
Plassering: Sentrert under tittellinjen.

Identitetsfeltet
Tag: platform
Callback:
```
brstr;
oh = findobj (gcf, 'Tag', 'Applybutt');
set (oh, 'Enable', 'on')
clear oh
```
Handling: Feltet brukes til å hente inn ønsket ID-nummer for nye plattformer, eller til å vise hvilken plattform som er under modifisering. I det første tilfellet er feltet editerbart og tomt, mens det i det andre tilfellet ikke er noen av delene. Når det gjelder callbacket, så sørger det for at brigadefeltet blir oppdatert (funksjonen

brstr () (3.8.5)), før Apply-knappen, som er disablet når det gjelder nye plattformer, gjøres tilgjengelig for interaksjon. Dette callbacket er aktivt også under modifisering, i tilfelle man ønsker å endre ID-nummeret. Tallet som vises ved modifisering hentes fra den globale *element* variabelen (3.6.3), fra feltet 'ID', og det er ID'en fra den plattformen som er markert i Plattformlisten i Element Control (3.5.1.2) som brukes.

Plassering: Den øverste editboksen. (Normalt et hvitt, innfelt rektangel.)

Typefeltet

Tag: platformtype

Callback: *addelementtype ()*

Handling: Feltet viser tilgjengelige plattformkategorier. Disse hentes fra variabelen *platformtypes* i filen *platformfile.mat* (3.7.2). Deretter kan brukeren velge hvilken type han vil ha, hvorpå callbacket aktiveres og oppdaterer Navnefeltet med de tilgjengelige plattformene av den gitte typen via funksjonen *addelementtype ()*.

Plassering: Andre editboks fra toppen.

Navnefeltets tekstboks

Tag: StaticText2

Callback: Ingen

Handling: Kun Ledetekst. 'NAME :' uansett bruk.

Plassering: Under Typefeltets tekstboks

Navnefeltet

Tag: platformname

Callback: Ingen

Handling: Bruker kan velge mellom de tilgjengelig plattformer av type som angitt i Typefeltet. Valget leses når man aktiverer Apply-knappen. Valgmulighetene hentes fra filen *platformfile.mat* (3.7.2). Om det er innholdet i variabelen *indirectnames*, *pbnames* eller *tanknames* som lastes, avhenger av hva som ble valgt i Typefeltet.

Plassering: Den nederste editboksen.

Brigadefeltet

Tag: brig

Callback: Ingen

Handling: Identifiserer brigadetilhørighet. Det som vises er i det store og hele bare det første sifferet i tallet fra Identitetsfeltet (tallet må være mellom 100 og 299 hvis dette skal virke).

Plassering: Nedenfor Navnefeltet.

Applyknappen

Tag: applybutt

Callback:

add = gcbf;		*mod = gcbf;*
PerformAddElement		*PerformModElement;*
figure (add)	eller	*figure (mod);*
clear add		*clear mod*
clrstr		*close gcbf*

Handling: Det er naturligvis her man trykker for å godkjenne de valg man har gjort i dialogboksen. Den første callback-sekvensen er aktiv ved innlegging av nye plattformer, og den andre ved modifisering av eksisterende. Den første kaller *PerformAddElement ()* (3.8.31), som oppretter en ny instans i den globale variabelen *elements* (3.6.3), samt henter inn typeavhengig tilleggsinformasjon fra

bruker. Deretter kalles *clrstr ()* (3.8.9) for å slette innholdet i Identitetsfeltet og Brigadefeltet, samt deaktivere Apply-knappen. Dialogboksen blir så stående åpen og aktiv, slik at flere plattformer kan defineres i samme slengen. Den andre sekvensen kaller *PerformModElement ()* (3.8.34), som gjør mye av det samme som *PerformAddElement (),* med det unntak at det ikke opprettes en ny plattform i variabelen *elements,* men letes frem til den som skal endres. Deretter lukkes vinduet.

Innlegging av den første plattformen medfører også aktivering (enabling) av Plattformlistens Modknapp, Plattformlistens Delknapp, Våpenlistens Addknapp, Observasjonsmiddellistens Addknapp og Plattformkommandolisten i Element Control.

Plassering: Nederst til venstre.

Avbrytknappen

Tag: okbutt
Callback: *global elhandle*
clear elhandle
close (gcbf);
Handling: Stenger vinduet uten å opprette flere nye plattformer, eventuelt uten å utføre endring på eksisterende plattform. Ved opprettelse av nye plattformer er dette den eneste måten å stenge vinduet på (hvis man ligger unna systemmenyene). Teksten på knappen gjenspeiler bruksområdet og er ’Close’ når Tittellinjen viser ’Add element’, og ’Cancel’ når tittellinjen viser ’Modify element’.
Plassering: Nederst til høyre.

3.5.2.2 Observasjonsmiddeldialogboksen

Filnavn: AddObsdevs
Tag: AddObsdevs
Returvariable: Bruker den globale strukturen *elements* (3.6.3)
Bruksområde: Denne dialogboksen brukes både ved innlegging av nye observasjonsmidler og ved endring av eksisterende. Den kalles alltid via en såkalt forfunksjon, og i dennes tilfelle er det to forskjellige som benyttes. Den første er *showaddobsdevs ()* (3.8.44). Denne funksjonen åpner dialogboksen og fyller inn data fra filer. Den andre funksjonen er *showmodobsdev ()* (3.8.49). I tillegg til å laste inn de samme data fra fil, endrer denne funksjonen ledetekstene og vindustittelen, slik at de blir tilpasset bruksområdet som er å modifisere et eksisterende våpen. Callbacket på Apply-knappen endres også.

Tittellinjen

Tag: Aksesseres som parameter til vinduet.
Callback: Ingen
Handling: Teksten skifter etter hvilken sammenheng vinduet åpnes i. Når kallet kommer fra *showaddobsdevs ()* (3.8.44) og hvis dialogboksen kalles direkte, er tittelen på vinduet ’New obs.dev.’. Når det åpnes fra *showmodobsdev ()* (3.8.49), endres tittelen til ’Modify obs.dev.’. Utover dette har ikke tittellinjen noen aktiv rolle.
Plassering: På toppen av vinduet.

Brukerinformasjon

Tag: StaticText1
Callback: Ingen
Handling: Forteller brukeren hva som forventes. For øvrig et helt bortkastet felt, ettersom tittellinjen kunne vært brukt til dette formålet, samt at brukeren som regel vet

hvorfor dialogboksen ble åpnet i utgangspunktet. Teksten skal være henholdsvis "Add obs.devs." eller "Modify obs.dev.".

Plassering: Sentrert under tittellinjen.

Identitetsfeltet

Tag: platform

Callback: Ingen

Handling: Feltet brukes til å vise hvilken plattform observasjonsmidlene relateres til. Feltet kan ikke editeres i noen tilfeller. Tallet som vises (plattform ID'en) hentes fra den globale *elements* variabelen (3.6.3), fra feltet 'ID', og det er ID'en til den plattformen som er markert i Plattformlisten i Element Control (3.5.1.2) som brukes.

Plassering: Den øverste editboksen.

Typefeltet

Tag: obsdevtype

Callback: *addobsdevtype ()*

Handling: Feltet viser tilgjengelige observasjonsmiddelkategorier. Disse hentes fra variabelen *obsdevtypes* i filen *obsdevfile.mat* (3.7.1). Det finnes foreløpig bare en kategori, men det er klargjort for flere på samme måte som i funksjonen *addweapontype ()* (3.8.4).

Plassering: Andre editboks fra toppen.

Navnefeltet

Tag: obsdevname

Callback: Ingen

Handling: Bruker kan her velge mellom de tilgjengelig observasjonsmidler. Valget leses når man aktiverer Applyknappen.

Plassering: Den nederste editboksen.

Applyknappen

Tag: applybutt

Callback:

add = gcbf;		*mod = gcf;*
PerformAddObsdev		*PerformModObsdev;*
figure (add)	eller	*figure (mod);*
clear add		*clear mod*
		close (gcbf)

Handling: Utfører de valg man har gjort i dialogboksen. Den første callback-sekvensen er aktiv for nye observasjonsmidler, og den andre ved modifisering. Den første kaller *PerformAddObsdev ()* (3.8.32), slik at et nytt observasjonsmiddel opprettes for den aktuelle plattformen i den globale variabelen *elements* (3.6.3). Dialogboksen blir så stående åpen og aktiv, slik at flere observasjonsmidler kan defineres i samme slengen. Den andre sekvensen kaller *PerformModObsdev ()* (3.8.35), som gjør mye det samme som *PerformAddObsdev ()*, med det unntak at det ikke opprettes et nytt observasjonsmiddel i variabelen *elements*, men letes frem til det som skal endres. Deretter lukkes vinduet.

Innlegging av det første observasjonsmiddelet for en gitt plattform medfører aktivisering av Observasjonsmiddellistens Modknapp, Observasjonsmiddellistens Delknapp og Observasjonsmiddelkommandolisten i Element Control.

Plassering: Nederst til venstre.

Avbrytknappen

Tag: okbutt

Callback: global elhandle
clear elhandle
close (gcbf);
Handling: Stenger vinduet uten å opprette eller modifisere observasjonsmidler. Teksten på knappen gjenspeiler bruksområdet og er ’Close’ når Tittellinjen viser ’New obs.dev’, og ’Cancel’ når tittellinjen viser ’Modify obs.dev.’.
Plassering: Nederst til høyre.

3.5.2.3 Våpendialogboksen

Filnavn: AddWeapons
Tag: AddElem
Returvariable: Bruker den globale strukturen *elements* (3.6.3)
Bruksområde: Denne dialogboksen brukes både ved innlegging av nye våpen og ved endring av eksisterende våpen. Den kalles alltid via en såkalt forfunksjon, og i dennes tilfelle er det to forskjellige som benyttes. Den første er *showaddweapons ()* (3.8.46). Denne funksjonen åpner dialogboksen og fyller inn data fra filer. Den andre funksjonen er *showmodweapon ()* (3.8.51), og den har noe mer å gjøre. I tillegg til å laste inn de samme data fra fil, endrer den ledetekstene og vindustittelen, slik at de blir tilpasset bruksområdet, som er å modifisere våpen. Callbacket til Apply-knappen endres også.

Tittellinjen
Tag: Aksesseres som parameter til vinduet.
Callback: Ingen
Handling: Teksten skifter etter hvilken sammenheng vinduet åpnes i. Når kallet kommer fra *showaddweapons ()* (3.8.46), eller hvis dialogboksen kalles direkte, er tittelen på vinduet ’New weapon’. Når det åpnes fra *showmodweapon ()* (3.8.51), endres tittelen til ’Modify element’. Utover dette har ikke tittellinjen noen aktiv rolle.
Plassering: På toppen av vinduet.

Brukerinformasjon
Tag: StaticText1
Callback: Ingen
Handling: Forteller brukeren hva som forventes. For øvrig et helt bortkastet felt, ettersom tittellinjen kunne vært brukt til dette formålet, samt at brukeren som regel vet hvorfor dialogboksen ble åpnet i utgangspunktet. Teksten skal være hhv "Add weapon" eller "Modify weapon".
Plassering: Sentrert under tittellinjen.

Identitetsfeltet
Tag: platform
Callback: Ingen
Handling: Feltet brukes til å vise hvilken plattform våpnene relateres til. Feltet kan ikke editeres i noen tilfeller. Tallet som vises (plattform ID’en) hentes fra den globale *element* variabelen (3.6.3), fra feltet ’ID’, og det er ID’en fra den plattformen som er markert i Plattformlisten i Element Control (3.5.1.2) som brukes.
Plassering: Den øverste editboksen.

Typefeltet
Tag: weapontype
Callback: *addweapontype ()*
Handling: Feltet viser tilgjengelige våpenkategorier. Disse hentes fra variabelen *weapontypes* i filen *weaponfile.mat* (3.7.4). Deretter kan brukeren velge hvilken

type han vil ha, hvorpå callbacket aktiviseres og oppdaterer Navnefeltet med de tilgjengelige våpnene av den valgte typen via funksjonen *addweapontype ()* (3.8.4). Det utføres ingen kontroll av hvorvidt et våpen logisk sett kan tilordnes en plattform av den aktuelle type.

Plassering: Andre editboks fra toppen.

Navnefeltet

Tag: weaponname
Callback: Ingen
Handling: Bruker kan velge mellom de tilgjengelig våpen av type som angitt i Typefeltet. Valget leses når man aktiverer Applyknappen.
Plassering: Den nederste editboksen.

Applyknappen

Tag: applybutt
Callback:

add = gcbf;		*mod = gcbf;*
PerformAddWeapon		*PerformModWeapon;*
figure (add)	eller	*figure (mod);*
clear add		*clear mod*
		close gcbf

Handling: Utfører de valg man har gjort i dialogboksen. Den første callback-sekvensen er aktiv for nye våpen, og den andre ved modifisering av eksisterende. Den første kaller *PerformAddWeapon ()* (3.8.33), slik at et nytt våpen opprettes for den aktuelle plattformen i den globale variabelen *elements* (3.6.3). Dialogboksen blir så stående åpen og aktiv, slik at flere våpen kan defineres i samme slengen. Den andre sekvensen kaller *PerformModWeapon ()* (3.8.36), som gjør mye av det samme som *PerformAddWeapon ()*, med det unntak at det ikke opprettes et nytt våpen i variabelen *elements*, men letes frem til det våpen som skal endres. Deretter lukkes vinduet.

Innlegging av det første våpen for en gitt plattform medfører aktivisering av Våpenlistens Modknapp, Våpenlistens Delknapp og Våpenkommandolisten i Element Control.

Plassering: Nederst til venstre.

Avbrytknappen

Tag: okbutt
Callback: global elhandle
clear elhandle
close (gcbf);
Handling: Stenger vinduet uten å opprette eller modifisere våpen. Ved opprettelse av nye våpen er dette den eneste måten å stenge vinduet på (hvis man ligger unna systemmenyene). Teksten på knappen gjenspeiler bruksområdet, og er ’Close’ når Tittellinjen viser ’New weapon’, og ’Cancel’ når tittellinjen viser ’Modify weapon’.
Plassering: Nederst til høyre.

3.5.2.4 Loggkategoriboksen

Fil: logsettingsdlg
Tag: Figure1
Returvariable: Status for de 6 avkrysningsboksene returneres som returverdi sammen med handle til vinduet (i følge ny GUIDE standard).

Bruksområde: Når det fra brukeren er ytret ønske om å opprette ny loggkategorifil, vil denne dialogboksen åpnes for å la brukeren spesifisere innholdet i den nye filen. Det finnes 6 avkrysningsbokser i dialogboksen, en for hver av de eksisterende loggkategorier.

Kategoriboksene

Alle dokumenteres under ett fordi de er så enkle.

Tag: checkdebug
checkdetection
checkmissile
checkfire
checkview
checkscreen

Callback: Følger den nye standarden til GUIDE

Handling: Brukes kun til å hente ja/nei-verdi for hver kategori.

Bekreftknappen

Tag: okbutt

Callback: Følger den nye standarden til GUIDE

Handling: Slipper programmet videre, noe som medfører returnering av returverdier. Legg merke til at det ikke finnes noe avbryt-alternativ.

3.5.2.5 Loggkategorifilboksen

Fil: logfiledlg

Tag: figure1

Returvariable: Det brukerspesifiserte filnavnet returneres til filen som kalte dialogboksen, sammen med en handle til vinduet og informasjon om hvilken knapp som ble valgt. Dette er en ekte returverdi.

Bruksområde: Vinduet åpnes når det er bedt om å opprette ny scenariofil ved lagring.

Filnavnfeltet

Tag: filename

Callback: Følger den nye standarden til GUIDE

Handling: Aktiviserer *bruk*-knappen når det finnes tekst i feltet.

Brukknappen

Tag: use

Callback: Følger den nye standarden til GUIDE

Handling: Lagrer navnet fra *filnavnfeltet* i scenariofilen uten å editere loggkategorifilen. Vær oppmerksom på at det ikke kontrolleres at filen finnes.

Opprettknappen

Tag: create

Callback: Følger den nye standarden til GUIDE

Handling: Åpner logsettingsdlg for å la bruker spesifisere loggkategorier. Navnet på filen lagres så på en slik måte at filen overskrives hvis den finnes, og opprettes hvis den ikke gjør det.

3.5.2.6 Scenariodialogboksen

Fil: NewScen

Tag: ns

Returvariable: De aktuelle data hentes ut fra dialogboksen direkte fra funksjonen *CreateScn ()* (3.8.14), som kalles fra callbacket til OK-knappen før dialogboksen lukkes.

Bruksområde: Når man har lagt inn alle plattformer, våpen og observasjonsmidler man ønsker, og gitt dem alle kommandoer som trengs, er det på tide å lagre opplysningene med tanke på å overføre dem til SIMBA Simulator. Det gjøres ved å bruke Savefilesknappen i Element Control. Denne kaller Lagredialogboksen (3.5.2.7) som inneholder bl.a. et valg for å opprette en ny Scenariofil (3.2). Hvis dette velges, vil Scenariodialogboksen (3.5.2.6) åpnes via *sfiles ()* (3.8.43). Så kan det legges inn verdier som definerer det nye scenariet.

Filfeltet

Tag: filename
Callback: Ingen
Handling: Viser navnet på scenariofilen. Dette navnet ble spesifisert av bruker i Lagredialogboksen (3.5.2.7).

Folderfeltet

Tag: result
Callback: Ingen
Handling: Lar bruker spesifisere hvilken katalog som skal brukes til å lagre resultater fra SIMBA Simulator. Katalogen opprettes på *$PRP-root$/result* hvis den ikke allerede finnes der.

Rundefeltet

Tag: Runs
Callback: Ingen
Handling: Lar bruker spesifisere hvor mange ganger simuleringen skal kjøres med de samme parametrene.

Startfeltet

Tag: Start
Callback: Ingen
Handling: Her kan man velge når simuleringen skal starte. Verdien representerer antall sekunder fra start. Den mest vanlige verdien er 0 (null).

Stegfeltet

Tag: Step
Callback: Ingen
Handling: I dette feltet angis hvilken tidsoppløsning SIMBA Simulator skal bruke. Det betyr at hvis dette feltet settes til 5, så kan plattformer bare detekteres hvert 5. sekund. De vil da også bare kunne skytes på hvert 5. sekund. Settes verdien for høyt, vil det bli rare resultater. Den vanligste verdien er 1. Da kjøres tiden bare 1 sekund mellom hver simuleringsrunde.

Stoppfeltet

Tag: Stop
Callback: Ingen
Handling: Forteller hvor lenge simuleringen skal kjøres. Oppgis i antall sekunder fra start. Det er her snakk om simulert tid, ikke realtime. Simuleringen stopper automatisk når alle plattformene har fullført alle sine kommandoer, så det er sterkt å anbefale at denne verdien settes slik at den i alle fall er stor nok.

OKknappen
Tag: PushButton1 (Samme som Avbrytknappen!)
Callback: *CreateScn;*
close(gcbf);
Handling: Kaller funksjonen som leser ut dataene fra dialogboksen, og lukker så vinduet. Bruk av denne knappen medfører opprettelse av en Scenariofil (3.2).
Avbrytknappen
Tag: PushButton1 (Samme som OKknappen)
Callback: *close (gcbf)*
Handling: Lukker dialogboksen uten å tolke innholdet.

3.5.2.7 Lagredialogboksen

Fil: SaveFile
Tag: Fig1
Returvariable: Overføres som argumenter til funksjonen *sfiles ()* (3.8.43).
Bruksområde: Kalles fra Element Control når Savefilesknappen betjenes, og lar brukeren spesifisere navn på scenariet. Som kjent får alle filene i Resultatfilsettet (3.2) samme fornavn. Det er dette fornavnet som spesifiseres her. Det kan også velges om det skal genereres en ny scenariofil eller ikke.

Filnavnfeltet
Tag: firstname
Callback: Ingen
Handling: Lar brukeren angi fornavnet til filene som skal genereres.

Avkryssingsboksen
Tag: newscn
Callback: Ingen
Handling: Når valget er merket, blir det opprettet en ny scenariofil. Dette gjøres ved at Scenariodialogboksen åpnes fra funksjonen *sfiles ()* (3.8.43), slik at brukeren kan spesifisere nødvendige data for scenariet. Verdien fra dette feltet overføres til funksjonen *sfiles ()* som parameter nummer 2.

Saveknappen
Tag: Pushbutton1
Callback:
```
global lhandle
lhandle = gcbf;
h = findobj (gcbf, 'Tag', 'firstname');
name = get (h, 'String');
h = findobj (gcbf, 'Tag', 'newscn');
new = get (h, 'Value');
sfiles (name, new);
delete(gcbf);
```
Handling: De to første linjene i callbacket er vel helt bortkastet, ettersom *gcbf* brukes som referanse til figuren både når den skal lukkes, og når data skal leses. Tredje til sjette linje leser verdiene fra Filnavnfeltet og Avkryssingsboksen, og linje syv kaller funksjonen *sfiles ()* (3.8.43) med disse verdiene som parametre. Linje åtte lukker dialogboksen. Kort sagt igangsetter knappen generering av Resultatfilsettet (3.2).

Avbrytknappen
Tag: Pushbutton2
Callback: *delete (gcbf)*

Handling: Lukker dialogboksen og avbryter lagringsprosessen.

3.5.2.8 Plattformhøydedialogboksen

Fil: linedata
Tag: Fig1
Returvariable: Data fra de to editboksene lagres i de globale variablene *vishobs* og *vizhtarg*.
Bruksområde: Når Visibility/Along line velges i hovedmenyen (3.5.1.1), kalles funksjonen *viz ()* (3.8.54), som ber om til/fra punkter på kartet i Hovedvinduet (3.5.1.1) før den kaller opp denne dialogboksen. Her kan brukeren spesifisere observasjonsmiddelhøyden til den skuende plattformen og høyden på den beskuede. Ved konfirmering kalles funksjonen *viz ()* igjen.

Skuehøydefeltet
Tag: hobs
Callback: Ingen
Handling: Tar i mot høyden til den plattformen som observerer terrenget og ser etter den andre plattformen. I praksis vil det være observasjonsmiddelets faktiske høyde over bakken som skal angis, og ikke plattformens fulle høyde.

Målhøydefeltet
Tag: htarg
Callback: Ingen
Handling: Tar i mot spesifikasjon av det antatte målets fulle høyde. Dette betyr at det holder å observere toppen av målet for å detektere det. Dette er samme antakelse som gjøres i simuleringsmodellen.

OKknappen
Tag: Pushbutton1
Callback:
```
global vizhobs vizhtarg vizoangle
hobshandle = findobj (gcbf, 'Tag', 'hobs');
svizhobs = get (hobshandle, 'String');
htarghandle = findobj (gcbf, 'Tag', 'htarg');
svizhtarg = get (htarghandle, 'String');
vizhobs = eval (svizhobs);
vizhtarg = eval (svizhtarg);
viz line_perform
delete(gcf)
```
Handling: Data lagres i globale variable før funksjonen *viz ()* (3.8.54) kalles. line_perform er egentlig en parameter, selv om det ikke ser slik ut for en ydmyk C-veteran ved første øyekast. Det antas videre at de to eval-linjene fungerer som en slags implisitt typekonvertering (fra char[] til int, på C språket). Poenget med variabelen *vizoangle* er fremdeles en gåte.

Avbrytknappen
Tag: Pushbutton1
Callback: *delete (gcf)*
Handling: Lukker dialogboksen.

3.5.2.9 Gjenopprettdialogboksen

Fil: loadel
Tag: Fig1
Returvariable: Bruker den globale *lhandle* til å overføre data tilbake til *loadelements ()* (3.8.26). Variabelen opprettes i callbacket.

Bruksområde: Valget Platforms/Load i hovedvinduet (3.5.1.1) utløser åpning av denne dialogboksen. Her kan bruker spesifisere hvilken element-fil som skal lastes, samt hvilket kart som ønskes brukt.

Elementfilfeltet
Tag: file
Callback: Ingen
Handling: Lar bruker skrive inn navet på Resultatfilsettet (3.2) som skal åpnes.

3.5.2.10 Kartfilfeltet

Tag: mapfile
Callback: Ingen
Handling: Brukes til angivelse av navn på Kartfilsett (3.3).

OKknappen
Tag: Pushbutton1
Callback: *global lhandle*
lhandle = gcbf;
loadelements;
delete (gcbf);
Handling: Oppretter en global variabel som fylles med en referanse til vinduet. Så kalles funksjonen *loadelements ()* (3.8.26) som bruker den globale variabelen til å aksessere dataene fra brukeren, deretter lukkes dialogboksen.

Avbrytknappen
Tag: Pushbutton2
Callback: *delete (gcbf)*
Handling: Avbryter gjenopprettingsprosessen og lukker dialogboksen.

3.5.2.11 Kartdialogboksen

Fil: loadmap
Tag: Fig1
Returvariable: Filnavnet overføres til funksjonen *loadall ()* (3.8.25) som parameter, men av en eller annen grunn legges det en referanse til dialogboksen i den globale variabelen *lhandle*. Denne referansen brukes til å lukke dialogboksen fra funksjonen *loadall ()*, i motsetning til det vanlige som er å lukke dialogboksen i selve callbacket.
Bruksområde: Denne dialogboksen kan trigges fra to steder. Enten fra menyvalget Kart/Load i Hovedvinduet (3.5.1.1), eller fra et klikk på ungløven Simba når SIMBA Preprosessor først startes. Uansett er det navnet på en kartfil som forventes. Det er verdt å legge merke til at det i denne dialogboksen er lagt callback på både Filnavnfeltet og OKknappen. Callback'ene er helt like, noe som fører til at knappene egentlig er overflødige.

Filnavnfeltet
Tag: mapfiletext
Callback: *message(' ');*
maphandle = findobj (gcbf, 'Tag', 'mapfiletext');
mapfile = get (maphandle, 'String');
global lhandle
lhandle = gcbf;
loadall (mapfile, 'con');

Handling: Brukeren forventes å spesifisere navnet på Kartfilsettet (3.3) som skal brukes i dette feltet. Så snart feltet mister fokus utføres callbacket, og noe av det første funksjonen *loadall ()* (3.8.25) gjør, er å lukke dialogboksen.

OKknappen
Tag: Pushbutton1 (Samme som Avbrytknappen)
Callback:
```
message(' ');
maphandle = findobj (gcbf, 'Tag', 'mapfiletext');
mapfile = get (maphandle, 'String');
global lhandle
lhandle = gcbf;
loadall (mapfile, 'con');
```
Handling: Dette callbacket er altså helt likt det for Filnavnfeltet. I praksis vil callbacket til OKknappen aldri bli kjørt, ettersom Filnavnfeltet vil utføre sitt så snart fokus flyttes derfra. Funksjonen *loadall ()* (3.8.25) kalles, og da lastes Kartfilsettet (3.3).

Avbrytknappen
Tag: Pushbutton1
Callback: *delete (gcbf)*
Handling: Lukker dialogboksen og avbryter lasting av Kartfilsett (3.3).

3.5.2.12 Tilleggsinfodialogboksen

Fil: platforminfowindow
Tag: InfoInputWindow
Returvariable: Kaller en egen funksjon som henter data før den selv lukker dialogboksen.
Bruksområde: Dette er den eneste multifunksjonelle dialogboksen i SIMBA Preprosessor-koden. Den brukes ved oppretting av plattformer til å hente tilleggsinformasjon fra brukeren om plattformen som opprettes. Hvilken informasjon som hentes avhenger av hvilken type plattform som opprettes. Tilleggsinfodialogboksen må alltid initieres via en forfunksjon, ettersom den ikke inneholder ledetekster (den vet jo ikke hva den skal hente). Hvis det er en direkteskytende plattform som er under opprettelse, vil det være høyden til plattformen og høyden til eventuelle siktbegrensende faktorer i startøyeblikket som hentes, men hvis det er en indirekteskytende plattform, vil det være antatt systemfeil i forbindelse med sikting (bom i x- og y-retning oppgitt som standardavvik). Uansett så lagres dataene i den globale *elements* strukturen (3.6.3). Boksen kalles fra både *PerformAddElement ()* (3.8.31) og *PerformModElement ()* (3.8.34).

Acceptknappen
Tag: Accept
Callback: *acceptinfo*
Handling: Kaller *acceptinfo ()* (3.8.1), en funksjon som leser ut data, lukker dialogboksen og lagrer data dit de skal.

3.5.2.13 Sektordialogboksen

Fil: secdata
Tag: Fig1
Returvariable: Bruker lokalt opprettede globale variable ved navn *vizhobs*, *vishtarg* og *vizoangle*.
Bruksområde: Når man skal undersøke synbarheten i en sektor fra et gitt punkt, brukes menyvalget Visibility/In sector i hovedmenyen, og da kommer denne dialogboksen sprettende.

Observatørfeltet
Tag: hobs
Callback: Ingen
Handling: Henter observatørens siktemiddelhøyde.

Målfeltet
Tag: htarg
Callback: Ingen
Handling: Henter det antatte målets høyde.

Vinkelfeltet
Tag: oangle
Callback: Ingen
Handling: Henter åpningsvinkelen til siktemiddelets linse. I praksis blir denne verdien ganget med 2, ettersom åpningsvinkelen anses å være vinkelen i forhold til "rett-frem".

OKknappen
Tag: Pushbutton1 (Samme som Avbrytknappen)
Callback:
```
global vizhobs vizhtarg vizoangle
hobshandle = findobj (gcbf, 'Tag', 'hobs');
svizhobs = get (hobshandle, 'String');
htarghandle = findobj (gcbf, 'Tag', 'htarg');
svizhtarg = get (htarghandle, 'String');
oanglehandle = findobj (gcbf, 'Tag', 'oangle');
svizoangle = get (oanglehandle, 'String');
vizhobs = eval (svizhobs);
vizhtarg = eval (svizhtarg);
vizoangle = eval (svizoangle);
viz sec_perform
delete(gcf)
```
Handling: Henter data fra dialogboksen og konverterer dem før de legges i globale variable. Deretter kalles funksjonen *viz ()* (3.8.54) med tekststrengen *sec_perform* som argument, og der nyttegjøres de globale variablene. Så stenges dialogboksen.

Avbrytknappen
Tag: Pushbutton1 (Samme som OKknappen)
Callback:
```
delete(gcf)
```
Handling: Lukker dialogboksen og avbryter siktbarhetstesten.

3.5.2.14 get_* dialogboksene

Her omtales dialogboksene som brukes for å kun hente en eller to parametre. Disse er stort sett så enkle at funksjonaliteten sier seg selv. Det som spesifiseres for den enkelte boks, begrenses derfor til filnavn, tagnavn, eventuelle callback, dataoverføringsmetode og triggerfunksjoner.

Ammoboksen
Fil: get_ammo
Tag: Fig3
Callback:
```
global  ammo
ammo = get (gcbo, 'String');
elcommand ('Ammo create_command');
close(gcbf);
```

Returvariable: Oppretter den globale variabelen *ammo* og lagrer data der.
Trigger: Når kommandoen Add_ammo gis til et våpen via Våpenkommandolisten i Element Control, kalles funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Ammoboksen, som kaller *elcommand ()* på nytt.

Angleboksen
Fil: get_angle
Tag: Fig3
Callback: *global vangle*
vangle = get (gcbo, 'String');
elcommand ('Angle create_command');
close(gcbf);
Returvariable: Oppretter den globale variabelen *vangle* og lagrer data der.
Trigger: Trigges fra valgene Direction og Sector fra Observasjonsmiddelkommandolisten, og Orientation fra Plattformkommandolisten. Samtlige kaller funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Angleboksen som kaller *elcommand ()* på nytt.

Coverboksen
Fil: get_cover
Tag: Fig3
Callback: *global vcover*
vcover = get (gcbo, 'String');
elcommand ('Cover create_command');
close (gcbf);
Returvariable: Oppretter den globale variabelen *vcover* og lagrer data der.
Trigger: Når kommandoen Cover gis til en plattform via Plattformkommandolisten i Element Control, kalles funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Coverboksen, som kaller *elcommand ()* på nytt.

Elevationboksen
Fil: get_elevation
Tag: Fig3
Callback: *global elevation*
elevation = get (gcbo, 'String');
elcommand ('elevation_create_command');
close (gcbf);
Returvariable: Oppretter den globale variabelen *elevation* og lagrer data der.
Trigger: Når kommandoen "Set Elevation" gis til et observasjonsmiddel via Observasjonsmiddelkommandolisten i Element Control, kalles funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Elevationboksen, som så kaller *elcommand ()* på nytt.

Flagboksen
Fil: get_flag
Tag: Fig3
Callback: *global flagname*
flagname = get (gcbo, 'String');
elcommand ('Flag_create_command');
close(gcbf);
Returvariable: Oppretter den globale variabelen *flagname* og lagrer data der.
Trigger: Når kommandoer vedrørende flagg gis til et element via kommandolistene i Element Control vinduet (3.5.1.2), kalles funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Flagboksen, som kaller *elcommand (…)* på nytt.

Hideboksen
Fil: get_hide
Tag: Fig3
Callback: *global hidetime*
hidetime = get (gcbo, 'String');
elcommand ('hide_create_command');
close(gcbf);
Returvariable: Oppretter den globale variabelen *hidetime* og lagrer data der.
Trigger: Når kommandoen Hide gis til en plattform via Plattformkommandolisten i Element Control, kalles funksjonen *elcommand (…)* (3.8.20), som igjen kaller opp Hideboksen, som kaller *elcommand (…)* på nytt.

Layersboksen
Fil: get_layers
Tag: Fig3
Callback: *global number_layers*
number_layers = get (gcbo, 'String');
elcommand ('StartArt create_command');
close(gcbf);
Returvariable: Oppretter den globale variabelen *ammo* og lagrer data der.
Trigger: Når kommandoen StartArt gis til et indirekteskytende våpen via Våpenkommandolisten i Element Control, kalles funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Layersboksen, som kaller *elcommand ()* på nytt.

Rangeboksen
Fil: get_range
Tag: Fig3
Callback: *global vrange*
vrange = get (gcbo, 'String');
elcommand ('Range create_command');
close(gcbf);
Returvariable: Oppretter den globale variabelen *vrange* og lagrer data der.
Trigger: Når kommandoen Range gis til et observasjonsmiddel via Observasjonsmiddelkommandolisten i Element Control, kalles funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Rangeboksen, som kaller *elcommand ()* på nytt.

Speedboksen
Fil: get_moveto_par
Tag: Fig3
Callback: *global speed*
speed = get (gcbo, 'String');
elcommand ('Move to create_command');
close(gcbf);
Returvariable: Oppretter den globale variabelen *speed* og lagrer data der.
Trigger: Når kommandoen *Moveto* gis til en plattform via Plattformkommandolisten i Element Control, kalles funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Speedboksen, som kaller *elcommand ()* på nytt.

Vegetationboksen
Fil: get_veg_height
Tag: Fig3
Callback: *global veg_height*

h = findobj (gcbf, 'Tag', 'Data');
veg_height = get (h, 'String');
close(gcbf);
makeveg

Returvariable: Oppretter den globale variabelen *veg_height* og lagrer data der.

Trigger: Kalles opp når menyvalget Vegetation/Make aktiviseres i Hovedmenyen. Denne dialogboksen har OK- og Cancelknapper. Derfor ligger ikke callbacket på editboksen, men på OKknappen. Cancelknappen har det vanlige callbacket (delete (gcbf)). Hvis Vegetationboksen lukkes ved hjelp av Okknappen, kalles funksjonen *makeveg ()* (3.8.29) som lar brukeren spesifisere hekkens utstrekning grafisk.

Waitboksen

Fil: get_wait
Tag: Fig3
Callback: *global waittime*
waittime = get (gcbo, 'String');
elcommand ('wait_create_command');
close(gcbf);

Returvariable: Oppretter den globale variabelen *waittime* og lagrer data der.

Trigger: Når kommandoen Wait gis til en plattform via Plattformkommandolisten i Element Control, kalles funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Waitboksen, som kaller *elcommand ()* på nytt.

Waitshotsboksen

Fil: get_waitshots
Tag: Fig3
Callback: *global waitshots*
waitshots = get (gcbo, 'String');
elcommand ('waitshots_create_command');
close(gcbf);

Returvariable: Oppretter den globale variabelen *waitshots* og lagrer data der.

Trigger: Når kommandoen Wait for shots gis til en plattform via Plattformkommandolisten i Element Control, kalles funksjonen *elcommand ()* (3.8.20), som igjen kaller opp Waitshotsboksen, som kaller *elcommand ()* på nytt.

3.6 Globale variable

Globale variable er variable som kan leses og skrives til fra hvor som helst i koden. I MatLab overgår deres livssyklus (lifecycle) selve programmets levetid, fordi alle globale variable holdes av MatLab selv, og ikke av programmet som kjøres. Programmet som sådan har ikke noe dedikert minneområde til egne variable, selv om funksjonene har det hver for seg. Det er dette som gjør at man kan opprette et par plattformer, stenge programmet og starte det igjen, og fremdeles ha tilgang alle plattformene fra forrige kjøring. Ønsker man å starte med blanke ark, eller et blankt kart i dette tilfellet, må man enten starte MatLab på nytt, eller slette de globale variablene manuelt. Innlasting av senario fra fil overskriver/sletter selvsagt gamle verdier.

Globale variable som kun brukes til transport av returverdier fra dialogbokser er ikke tatt med i dette kapittelet. De er omtalt i beskrivelsen av den enkelte dialogboks.

3.6.1 cmdhandle

Som navet antyder er dette en referanse; et såkalt håndtak (handle). Det som det refereres til er Element Control vinduet, og variabelen brukes stort sett til å finne objekter i vinduet ved hjelp

av deres tag'er. En typisk bruksmåte er *findobj (cmdhandle, 'Tag', 'platformlist')*, som ville hentet en referanse til plattformlisten i vinduet, slik at denne kan manipuleres. Variabelen settes flere steder i koden, men alltid til å peke til Element Control vinduet.

3.6.2 dx/dy

Disse variablene initieres i funksjonen *ld ()* (3.8.22), der de settes lik tilsvarende verdier fra Kartdatafilen (3.3.4). De representerer avstanden mellom punktene i høydematrisen i Terrengfilen (3.3.2), henholdsvis i øst/vest- og nord/sør-retning. I Kartdatafilen er avstandene oppgitt i desimeter, men i *ld ()* blir de delt på 10 og lagres således i meter. Variablene nyttes i funksjonene *CB ()* (3.8.6), *drawelement ()* (3.8.16), *showmap ()* (3.8.48) og *viz ()* (3.8.54), og i tillegg er de helt bortkastet deklarert i *drawrectangle ()* (3.8.19), *li ()* (3.8.23) og *lm ()* (3.8.24).

3.6.3 elements

Dette er kanskje den mest sentrale variabelen i SIMBA Preprosessor. I denne lagres all informasjon om elementene som brukeren spesifiserer. Unntaket som bekrefter regelen er vegetasjonshekkene (i den grad de kan anses som elementer) som lagres i en egen temporær fil. Strukturen er enkel nok for alle med bakgrunn i C; et *array* av *struct'er*. Hva det spesifikt kalles i MatLab, er noe usikkert, men så lenge det er enkelt å aksessere informasjonen, spiller det vel heller ikke så stor rolle. *Struct'ene* som bygger opp *arrayet* (eller matrisen) består av følgende felter:

- ✓ ID : Lagrer plattformens identifikasjonsnummer. Flere kan ha samme nummer, uten at det er sikkert at SIMBA Simulator godtar det. Feltet er en tallverdi.
- ✓ TYPE : Plattformens type. Informasjon fra både Typefeltet og Navnefeltet i Plattformdialogen lagres her som tekst.
- ✓ WEAPONS : Oversikt over hvilke våpen som er tilordnet plattformen. Informasjon om våpentype , så vel som våpennavn lagres her. Feltet består av en tekststreng for hvert våpen.
- ✓ NWEAPONS : Angir hvor mange variable som finnes lagret i WEAPONS. Altså hvor mange våpen som er tilordnet plattformen
- ✓ BRIG: Angir hvilken brigade plattformen tilhører. Dette er også et tallfelt.
- ✓ NOBS_DEVS : Holder rede på antallet observasjonsmidler som er lagret i OBS_DEVS. Dette feltet er for OBS_DEVS hva NWEAPONS er for WEAPONS. Inneholder en tallverdi.
- ✓ OBS_DEVS : Her lagres selve observasjonsmidlene. Både type og navn lagres i en tekst-variabel, og det lagres en tekststreng for hvert observasjonsmiddel.
- ✓ NCOM : Antall kommandoer i COMMANDS. NCOM er for COMMANDS hva NWEAPONS er for WEAPONS
- ✓ COMMANDS : Her stappes det inn en tekstvariabel for hver kommando.
- ✓ POS : Holder rede på hvor plattformer er etter siste kommando. Ved sletting av kommandoer mister denne variabelen lett tråden. Den kan synkroniseres ved bruk av enten Cleanupknappen eller Redrawknappen i Element Control.
- ✓ DIR : Her holdes informasjon om plattformens retning. Dette brukes for å beregne om et missil treffer plattformen forfra, bakfra eller fra siden.
- ✓ INFO1 : Lagrer tilleggsinformasjon for plattformen. Det som legges her kommer direkte og uprosessert fra Tilleggsinfodialogboksen (3.5.2.12), og lagres som tekst.
- ✓ INFO2 : Som INFO1.

Når det er opprettet flere av disse *struct'ene* og en av dem har fått flere våpen, kan for eksempel det siste våpenet til plattform nummer to i listen aksesseres slik:

våpen = elements {2}.WEAPONS {elements {2}.NWEAPONS};

Variabelen elements opprettes første gang i funksjonen *readelem ()* (3.8.39), skjønt ettersom de globale variablene jo lever fra kjøring til kjøring, er det vel kanskje riktigere å si at den initieres der. Imidlertid initieres den kanskje andre steder i koden også. Denne variabelen brukes av så godt som alle funksjonene i SIMBA Preprosessor.

3.6.4 fname

Verdien på denne variabelen brukes fra *sfiles ()* (3.8.43), der den initieres, og fra *CreateScn ()* (3.8.14). Det er fornavnet på Resultatfilsettet (3.2) som lagres her, og det hentes fra Lagredialogboksen (3.5.2.7).

3.6.5 IMG

Her lagres selve bildet fra Kartbildefilen (3.3.1), som lastes i *li ()* (3.8.23). Bildet brukes kun fra *showmap ()* (3.8.48).

3.6.6 insertbefore

Denne variabelen anga i sin tid om en ny kommando skulle lagres før eller etter den som var markert i kommandolisten i Element Control. Den er ikke lenger i bruk, men den er fremdeles deklarert og brukt i *elcommand ()* (3.8.20), *insertcommand ()* (3.8.20.2) og *loadelements ()* (3.8.26). Verdien er alltid null (false) slik at man alltid setter inn den nye kommandoen etter gjeldende kommando.

3.6.7 isconplot

Her har vi en boolsk variabel (TRUE/FALSE) som angir om kartet skal vises som contourplot eller ikke. Den settes i *showmap ()* (3.8.48), og leses fra *CB ()* (3.8.6) og *viz ()* (3.8.54).

3.6.8 M

Denne variabelen holder en kopi av dataene fra Terrengfilen (3.3.2). Den initialiseres i *lm ()* (3.8.24), og brukes fra *showmap ()* (3.8.48) og *viz ()* (3.8.54). I tillegg er den deklarert unødvendig i *CB ()* (3.8.6). Nevner også at det finnes en lokal variabel med samme navn i *editFOF2 ()* , men den har ingenting med dette å gjøre. Den har samme navn bare for å forvirre.

3.6.9 M2

Dette er en arbeidskopi av *M*. Den inneholder altså en modifisert versjon av dataene fra Terrengfilen. Den initialiseres og brukes i *showmap ()* (3.8.48), og den brukes også i *conplt ()* (3.8.10).

3.6.10 mainhandle

Som *cmdhandle* (3.6.1) er dette et håndtak til et vindu, nemlig Hovedvinduet (3.5.1.1). Bruksmåten er den samme, men i praksis brukes nok dette håndtaket mest til å få tak i kartet, eller til å aktivere Hovedvinduet, slik at man kan se og tegne på kartet. *mainhandle* settes mange steder (nesten hver gang man klikker på noe i Hovedvinduet), og den brukes fra *CB ()* (3.8.6), *cleanup ()* (3.8.8), *editFOF2 ()*, *elcommand ()* (3.8.20), *elplot ()* (3.8.21), *loadall ()* (3.8.25), *message ()* (3.8.30) og *wallplot ()* (3.8.56)

3.6.11 mapname

Her lagres navnet på det aktive Kartfilsettet (3.3). Variabelen initialiseres i *loadall ()* (3.8.25), og brukes fra *CreateScn ()* (3.8.14) og *showmap ()* (3.8.48)

3.6.12 nelem

Her har vi den gode, gamle telleren. Variabelen *nelem* er en heltallsvariabel så god som noen, og det den teller er antallet *struct'er* som er lagt inn i variabelen *elements* (3.6.3). Den brukes således i alle funksjoner som bruker elements, og det er som sagt de fleste.

3.6.13 nwall

Her er variabelen som holder rede på om det er gjort endringer vedrørende vegger på kartet (walls). Den teller også hvor mange vegger som er lagt til. Variabelen initialiseres i *lwall ()* (3.8.28), inkrementeres i *makeveg ()* (3.8.29) og brukes fra *swall ()* (3.8.52).

3.6.14 nx/ny

Her lagres antall punkter i henholdsvis nord/syd- og øst/vest-retning; antall punkter i Terrengfilen (3.3.2) altså. Verdiene leses direkte fra Kartdatafilen (3.3.4) via *ld ()* (3.8.22). Videre brukes de fra *CB ()* (3.8.6), *lm ()* (3.8.24), *lv ()* (3.8.27) og *viz ()* (3.8.54). Også disse variablene er deklarert en del steder der de likevel ikke benyttes, blant annet i *showmap ()* (3.8.48).

3.6.15 plottype

Denne variabelen holder rede på hvordan brukeren har bedt om å få kartet tegnet. Den settes i *showmap ()* (3.8.48), og både brukes og settes i *CB ()* (3.8.6).

3.6.16 skx/sky

Her holdes punktavstanden på et eventuelt zoomet kart. Verdien settes både fra *CB ()* (3.8.6) og *ld ()* (3.8.22), samt at den benyttes fra *showmap ()* (3.8.48)

3.6.17 snx/sny

Disse verdiene settes i utgangspunktet lik henholdsvis *nx* og *ny*, og brukes til å lagre antall punkter på kartet når det er zoomet. Verdien settes i både *ld ()* (3.8.22) og *CB ()* (3.8.6), samt at den brukes fra *drawelement ()* (3.8.16), *showmap ()* (3.8.48) og *viz ()* (3.8.54).

3.6.18 sx/sy

Inneholder kopi av *dx*/*dy* for bruk ved zooming. Verdien settes i både *ld ()* (3.8.22) og *CB ()* (3.8.6), samt at den brukes fra *showmap ()* (3.8.48) og *viz ()* (3.8.54).

3.6.19 utmx0/utmy0

Variablene angir henholdsvis kartets sydligste og vestligste punkt, og initialiseres i *ld ()* (3.8.22) med verdi fra Kartdatafilen (3.3.4). Brukes fra *drawindirekt ()* (3.8.18), *editFOF2 ()*, *elcommand ()* (3.8.20), *elplot2 ()* (3.8.21), *makeveg ()* (3.8.29), *readcom ()* (3.8.38), *showmap ()* (3.8.48) og *wallplot ()* (3.8.56).

3.6.20 utmzone

Denne variabelen lagrer en tilsvarende verdi fra Kartdatafilen (3.3.4) som ikke brukes til noe. Den initialiseres i *ld ()* (3.8.22).

3.6.21 V

Her lagres verdiene fra Vegetasjonsfilen (3.3.3). Hele filen leses inn i *lv ()* (3.8.27), og verdiene brukes fra *showmap ()* (3.8.48) og *viz ()* (3.8.54).

3.6.22 viewpoint

Denne variabelen styrte i sin tid hvor man så kartet fra i et 3D perspektiv. Den er ikke lenger i bruk, og står fast til [0 80]. Den initialiseres i *loadall ()* (3.8.25), og refereres fra *showmap ()* (3.8.48).

3.6.23 W

Dette er en variabel som kun brukes i *showmap ()* (3.8.48). Den hadde således ikke trengt å være global.

3.6.24 x2/y2

Disse variablene lagrer verdier derivert fra *dx/dy* (3.6.2) og *snx/sny* (3.6.17). De initialiseres og brukes flittig i *showmap ()* (3.8.48), og brukes deretter bare fra *conplt ()* (3.8.10).

3.7 Datasett på fil

I noen tilfeller er det praktisk å lagre data på eksterne filer. Dette gjør det enklere å endre innholdet, fordi man ikke trenger å gå inn i kildekoden. Slike filer lagrer gjerne data digitalt. I SIMBA Preprosessor finnes 6 slike filer, og for hver av dem angis det her hvilke data-entiteter de inneholder og hva disse er satt til. I tillegg antydes det hva de forskjellige entitetene brukes til.

3.7.1 obsdevfile.mat

Inneholder to datasett som brukes til å fylle listene i Observasjonsmiddeldialogboksen (3.5.2.2):
obsdevtypes = 'EYE'
eyenames = 'eye' 'ir'
Innholdet fra *obsdevtypes* fylles inn i Typefeltet, og innholdet fra *eyenames* listes i Navnefeltet.

3.7.2 platformfile.mat

Inneholder 4 datasett som brukes til å fylle listene i Plattformdialogboksen (3.5.2.1):
platformtypes = 'PB' 'TANK' 'INDIRECT'
tanknames = 'T80' 'BMP' 'LEO'
pbnames = 'JAVELIN' 'TOW' 'CV90'
indirectnames = 'ARTILLERY'
platformtypes settes inn i Typefeltet, og *pbnames* i Navnefeltet. Når man så endrer valget i Typefeltet, byttes innholdet i Navnefeltet ut med innholdet i en av de resterende datasettene, alt etter hva som velges.

3.7.3 SYMBX.sym og SYMBY.sym

Inneholder data for tegning av plattformsymboler. I praksis tegnes en haug med streker i forskjellige vinkler som all krysser hverandre på midten. De skulle i de fleste tilfeller resultere i en sirkel.

3.7.4 weaponfile.mat

Inneholder 5 datasett som brukes til å fylle listene i Våpendialogboksen (3.5.2.3):

weapontypes = 'GUN' 'FF' 'ART'
gunnames = '125mmHEAT' '125mmREFLEKS' '30mmAPDS'
ffnames = 'javelin' 'javelin2' 'tow_2a'
konvartnames = 'konvart'
smartartnames = 'smartart'
weapontypes settes inn i Typefeltet, og *gunnames* i Navnefeltet. Når man så endrer valget i Typefeltet, byttes innholdet i Navnefeltet ut med innholdet i en av de resterende datasettene, alt etter hva som velges. Dette er i praksis en liste over våpeneffektfiler.

3.8 Funksjoner

I dette kapittelet skal alle kodefilene dokumenteres. Beskrivelsen vil følge filnavnet. I de tilfellene der det i tillegg finnes funksjonsdeklarasjoner lokalt i filer, altså tilsvarende subfunksjoner, vil disse bli omtalt under den fil de ligger i. Med unntak av disse subfunksjonene, kalles alle funksjoner ved å referer til navnet på filen og ikke funksjonen.

Alle kodefilene skal nå være brukbart kommentert, derfor gjentas ikke koden i dette kapittelet. Det gis bare en oversikt over ansvarsområder og eventuelle ting av spesiell interesse. Funksjonene kommer i alfabetisk rekkefølge uten hensyn til store bokstaver. Filnavnene er identisk med funksjonsnavnene hvis ikke annet er angitt.

3.8.1 acceptinfo ()

Dette er callbacket til Acceptknappen i Tilleggsinfodialogboksen. Det funksjonen gjør er bare å legge verdiene fra de to feltene inn i *elements* (3.6.3), henholdsvis i feltene INFO1 og INFO2. De globale variablene som brukes er *cmdhandle* og *nelem* (deklarert, men ikke referert), i tillegg til *elements*.

3.8.2 addelementtype ()

Denne funksjonen blir kalt fra callbacket til Typefeltet i Plattformdialogboksen. Den bruker ingen globale variable og kaller ingen andre funksjoner, men den laster innholdet i filen *platformfile.mat* (3.7.2). Det som gjøres er å laste informasjon fra filen inn i Navnelisten i samme dialogboks. Hvilken variabel fra filen (*indirectnames*, *pbnames* eller *tanknames*) som lastes, avhenger av hva som ble valgt i Typelisten.

3.8.3 addobsdevtype ()

Samme som for funksjonen *addelementtype ()* (3.8.2), men for Observasjonsmiddeldialogboksen (3.5.2.2). Filen som lastes er *obsdevfile.mat* (3.7.1).

3.8.4 addweapontype ()

Samme som *addelementtype ()* (3.8.2), men for Våpendialogboksen (3.5.2.3). Filen som lastes her er *weaponfile.mat* (3.7.4).

3.8.5 brstr ()

Denne funksjonen kalles fra callbacket til Identitetsfeltet i Plattformdialogboksen. Den har som eneste oppgave å kopiere første siffer i plattformID'en ned til Brigadefeltet. Til det trenger den verken globale variable elle andre funksjoner.

3.8.6 CB (action)

Dette er funksjonen som kalles fra menyvalgene under Map/Zoom. Parameteren *action* angir hvilket menyvalg som ble foretatt, og variabelen brukes i en switch/case konstruksjon. Denne

koden har vært lite brukt, ettersom man ikke fant noe særlig behov eller nytteverdi i denne funksjonaliteten. Det er derfor ikke sikkert at koden er vanntett, for å si det mildt.

Funksjonen bruker de globale *plottype* (3.6.15) og *mainhandle* (3.6.10). Den første angir hvordan kartet skal tegnes, og den andre angir hvor det skal tegnes. I tillegg opprettes det en rekke globale variable lokalt, som ingen lenger kan gjøre rede for nytten av. De fleste later til å brukes fra funksjonen *showmap ()* (3.8.48), altså når kartet skal tegnes.

3.8.7 changeplatform ()

Dette er funksjonen som blir kalt når det velges en plattform i Plattformlisten i Element Control. Den sørger for å oppdatere status på knappene under Plattformlisten, og den setter det første elementet aktivt i samtlige kommandolister. Etter den er ferdig, kaller den *platforminfo* for å få oppdatert våpen- og observasjonsmiddellisten, og deres knapper. Denne funksjonen kalles også når en plattform slettes. De vanlige globale variablene brukes; *elements* (3.6.3), *nelem* (3.6.12) og *cmdhandle* (3.6.1).

3.8.8 cleanup ()

Denne funksjonen kalles når brukeren aktiviserer Cleanupknappen i Element Control. Den sørger for å tegne kartet på nytt, med alle plattformer tegnet kun i deres siste kjente posisjon (i følge kommandolisten). Denne posisjonen skal være lagret i feltet *POS* i *elements*, og den leses derfor derfra. Underveis kalles *showmap()* (3.8.48), *readwords ()* (3.8.40), *drawindirect ()* (3.8.18) og *drawelement ()* (3.8.16), litt avhengig av hvilke plattformer som finnes. Globale variable som benyttes er *elements* (3.6.3), *nelem* (3.6.12) og *mainhandle* (3.6.10).

3.8.9 clrstr ()

Tømmer Identitetsfeltet og Brigadefeltet i Plattformdialogboksen, samt disabler Applyknappen. Applyknappen er selv trigger for denne funksjonen, som ikke bruker verken globale variable eller andre funksjoner.

3.8.10 conplt (confile)

Denne funksjonen forventer å få navnet på et Kartfilsett (3.3) som parameter (*confile*). Hvis det finnes en Matrisefil (3.3.5) til Kartfilsettet, lastes denne før *contour2 ()* (3.8.11) kalles. Hvis ikke, lages det en slik fil, og kartet tegnes. Kallet kommer fra OKknappen i Kartdialogboksen eller fra *showmap ()* (3.8.48), og de globale variable som benyttes omfatter *x2* (3.6.24), *y2* (3.6.24), *M2* (3.6.9) og *W* (3.6.23). Som navnet tilsier, er det contourplot det dreier seg om her.

3.8.11 contour2 (c)

Denne funksjonen har noe så sjeldent som returvariable. Den returnerer tre verdier tilbake til den funksjonen den ble kalt fra. Det later imidlertid ikke til at disse verdiene tas imot noe sted, men de kan kanskje være nyttige til debugging. Det denne funksjonen gjør er å tegne contourplottet fra en såkalt Matrisefil (3.3.5). Parameteren *c* er en verdi som ble hentet fra matrisefilen i funksjonen som kalte *contour2 ()*. I matrisefilen heter den imidlertid *C* (stor c). Ingen funksjonskall eller globale variable er i bruk her.

3.8.12 convert (num)

Denne funksjonen brukes til å konvertere en tallverdi (*num*) til en streng, samt å putte inn et mellomrom fremst i den nye strengen. Grunnen til det siste er at MatLab fjerner såkalte ”trailing blanks”, altså mellomrom på slutten av en streng, når den setter sammen flere strenger til en. Noe som kan være svært plagsomt når man prøver å bygge strenger der det er mellomrom som

skiller verdier. For eksempel vil *strcat ("hei ", " ", "du");* resultere i en streng med 5 bokstavtegn uten mellomrom. Kalles fra svært mange steder i koden. Også denne funksjonen har returverdi. Bare én riktignok, men denne brukes til gjengjeld. Det er den resulterende strengen som returneres.

3.8.13 copycurrent ()

Bruker *elements* (3.6.3) og *nelem* (3.6.12) i sitt arbeid med å kopiere den aktive plattform. Kallet kommer fra Kopieringsknappen i Element Control. Denne funksjonen rett og slett kopierer en hel instans fra *elements*, med kommandoer og våpen og det hele, og legger den til på enden av arrayet.

3.8.14 CreateScn ()

Denne funksjonens funksjon er, ikke uventet, å opprette en scenariofil (3.2). Kallet kommer fra Scenariodialogboksen (3.5.2.6), og funksjonen henter ut data fra denne og lagrer dem til fil. Dialogboksen logfiledlg åpnes for å hente inn navn på filen som skal brukes til å spesifisere hvilke logg-grupper som skal skrives fra SIMBA Simulator, og hvis file ikke eksisterer (eller bruker ber om å overskrive), åpnes dialogboksen logsettingsdlg, slik at bruker kan spesifisere kategorier. I tillegg forsøkes det opprettet en katalog for loggfilene fra SIMBA Simulator, som angitt i dialogboksen. Vær obs på at denne funksjonen faktisk beveger seg rundt i forskjellige kataloger, noe som betyr at hvis programmet skulle bli avbrutt mens denne funksjonen er aktiv, er det ikke mulig å si noe sikkert om hva som er aktiv katalog i MatLab etterpå. Dette fordi det er MatLab sin fokus som flyttes, ikke bare funksjonens.

Funksjonen bruker de globale variablene *fname* (3.6.4) og *mapname* (3.6.11). Den første er navnet på scenariet som ble spesifisert i Lagredialogboksen (3.5.2.7), og den andre er navnet på kartsettet som ble angitt i Kartdialogboksen (3.5.2.11).

3.8.15 delcell (cellin, del)

Denne funksjonen skulle egentlig slette en indeks i en matrise. Det er slett ikke sikkert den gjør det. På den andre siden skader det ikke at ubrukte indekser blir liggende bakerst i matrisene slik programmet er skrevet. Det kalles bare hit en gang, og det er fra linje 579 i *elcommand ()* (3.8.20). Denne funksjonen er helt overflødig ettersom den funksjonalitet den var tiltenkt kan utføres langt mer elegant og hensiktsmessig.

3.8.16 DeleteElement (arg)

Denne funksjonen kalles fra tre steder. Hver av de tre listeboksene øverst i Element Control (3.5.1.2) har en Delknapp, og samtlige av disse kaller denne funksjonen. Det betyr at funksjonens ansvar er å slette et element i henhold til hvor kallet kommer fra og hva som er markert i den aktuelle listen. Det brukes en enkel switch/case basert på arg, som er et tall, for å rute funksjonaliteten. Filen inneholder tre subrutiner:

3.8.16.1 removeplatform (n)

Sletter plattform nummer *n* i den globale variabelen *elements* (3.6.3). Parameteren representerer nummeret i Plattformlisten og ikke ID-nummeret. Altså representerer det også indeksnummeret i *elements* (3.6.3). I tillegg sørges det for at Element Control (3.5.1.2) oppdateres i henhold til den plattformen som overtar fokus.

3.8.16.2 removeweapon (n, p)

Sletter det markerte våpenet for den markerte plattformen. Parameteren *n* representerer plattformens indeks, og *p* representerer våpenets indeks. Deretter vurderes status for Våpenkommandolisten med tilhørende knapper i forhold til om det finnes flere våpen eller ikke.

3.8.16.3 removeobsdev (n, p)

Sletter observasjonsmiddelet med indeks *p* fra plattformen med indeks *n*. Også her vurderes status for listen og dens knapper.

3.8.17 drawelement (x, y, rot, element)

Her har vi funksjonen som tegner de direkteskytende plattformene i Kartområdet i Hovedvinduet. Parametre: *x* og *y* er plattformens posisjon, *rot* er en rotasjonsvinkel som ikke brukes lenger (ettersom alle direkteskytende plattformer tegnes som sirkler, blir det meningsløst å rotere dem), og element representerer plattformen som skal tegnes. Igjen er det indeksnummeret i *elements* som brukes som referanse. Globale variable som benyttes er *snx* (3.6.17), *sny* (3.6.17), *dx* (3.6.2), *dy* (3.6.2) og *elements* (3.6.3), og det kalles til funksjonen *symbol ()* (3.8.53) underveis. Kall kan komme fra *cleanup ()* (3.8.8), *elcommand ()* (3.8.20) og *elplot2 ()* (3.8.21).

3.8.18 drawindirect (centerX, centerY, width, depth, angledeg, index)

Dette er en funksjon for utregning av hjørnene til nedslagsfeltene til de indirekteskytende plattformene. Parametrene; *centerX* og *centerY* er midtpunktet i nedslagsfeltet, *width* og *depth* angir utstrekningen, *angledeg* angir vridningen i grader, og *index* er plattformens indeksnummer i *elements* (3.6.3). De globale variablene *utmx0* (3.6.19) og *utmy0* (3.6.19) brukes for å finne det faktiske midtpunktet på skjermen, så regnes hjørnekoordinatene ut før *drawrectangle ()* (3.8.19) kalles for å faktisk tegne rektangelet. Ettersom atrilleri ikke lenger oppgis som nedslagsfelt, betyr dette i praksis at alle hjørnene er like.

3.8.19 drawrectangle (A , B, C, D, elements)

Tegner et rektangel som representerer et nedslagsfelt på kartet. Parametrene *A*, *B*, *C* og *D* angir hjørnene, og *element* er fremdeles plattformens indeksnummer i *elements* (3.6.3). Hjørnene regnes ut i *drawindirect ()* (3.8.18), som kaller denne funksjonen. Globale variable som anvendes er *snx* (3.6.17), *sny* (3.6.17), *dx* (3.6.2), *dy* (3.6.2) og *elements* (3.6.3). Ettersom alle hjørnene nå ligger i siktepuktet, tegnes det bare en prikk (et veldig lite rektangel). drawrectangle(A, B, C, D, elements)

3.8.20 elcommand (list, index)

Her har vi en av de mer sentrale funksjonene. Den er callback for alle tre rullgardinmenyene i kommandoområdet i Element Control (3.5.1.2), og også for svært mange av dialogboksene med filnavn av typen get_*.m. Det er her kommandoer legges til i kommandolisten til plattformene. Parametrene er *list* som angir hvor kallet kommer fra, og *index* som angir indeksnummeret dersom kallet kommer fra en av kommandolistene. Hvis *list* er et heltall fra 1 til og med 3, tolkes det som om kallet kom fra en kommandoliste. Hvis ikke, skal *list* være en tekststreng, og da brukes ikke *index*. Et lite hav av globale variable brukes; *elements* (3.6.3), *cmdhandle* (3.6.1), *mainhandle* (3.6.10), *utmx0* (3.6.19), *utmy0* (3.6.19), *insertbefore* (3.6.6), pluss en haug med transportvariable som både opprettes og forsøkes slettet lokalt i filen.

3.8.20.1 arctan (a, b)

Subrutine i filen elcommand.m, som er en versjon av MatLabs egen *atan ()* funksjon, men med sterkere fokus på kvadrant plassering. Regner ut arcustangens og justerer for 360 graders vinkel.

3.8.20.2 InsertCommand (command)

Funksjon for å stappe en streng inn i en liste av strenger. Mer spesifikt, en kommandostreng inn i den aktive plattforms kommandoliste. Skjermen oppdateres ikke. Globale variable i bruk er *cmdhandle* (3.6.1), *elements* (3.6.3) og *insertbefore* (REF). Kalles naturligvis kun fra *elcommand ()*. Parameteren command inneholder tekststrengen som representerer kommandoen.

3.8.21 elplot ()

Denne funksjonen ligger i filen elplot2.m. Kall til funksjonen vil altså være *elplot2 ()*. Ansvarsområdet omfatter igangsetting av tegning av samtlige plattformer på kartet, i alle kjente posisjoner. For å utføre selve tegningen kalles *drawelement ()* (3.8.16) eller *drawindirect ()* (3.8.18). I tillegg kalles funksjonen *message ()* (3.8.30) for å oppdatere Meldingsområdet i Hovedvinduet (3.5.1.1). Globale variable; *elements* (3.6.3), *nelem* (3.6.12), *mainhandle* (3.6.10), *utmx0* (3.6.19) og *utmy0* (3.6.19). Kallet kommer enten fra Omtegningsknappen eller fra *loadelements ()* (3.8.26).

3.8.22 ld (datafile)

Leser data fra fil. Det er Kartdatafilen (3.3.4) som hentes inn av denne funksjonen. Kalles fra funksjonen *loadall ()* (3.8.25). Globale variable; *ny* (3.6.14), *nx* (3.6.14), *dx* (3.6.2), *dy* (3.6.2), *sx* (3.6.18), *sy* (3.6.18), *snx* (3.6.17), *sny* (3.6.17), *skx* (3.6.16), *sky* (3.6.16), *utmx0* (3.6.19), *utmy0* (3.6.19) og *utmzone* (3.6.20). Kaller kun *message ()* (3.8.30), og det er for å holde bruker oppdatert via Meldingsområdet under kartet i Hovedvinduet (3.5.1.1).

3.8.23 li (imgfile)

Også her er det å hente data fra fil som står på programmet. Denne gangen er det Kartbildefilen i Kartfilsettet (3.3.1), altså det digitale kartet. Kallet kommer fra *loadall ()* (3.8.25), og de globale variable som er deklarert er; *IMG* (3.6.5), *ny* (3.6.14), *nx* (3.6.14), *dx* (3.6.2), *dy* (3.6.2), *sx* (3.6.18), *sy* (3.6.18), *snx* (3.6.17), *sny* (3.6.17), *skx* (3.6.16) og *sky* (3.6.16). Selve lesingen foregår via kall til en av MatLabs interne funksjoner.

3.8.24 lm (mapfile)

Igjen er lesing fra fil ansvarsområde, denne gangen fra Terrengfilen (3.3.2). Kallet kommer også her fra funksjonen *loadall ()* (3.8.25) og de globale variablene som benyttes er; M (3.6.8), *ny* (3.6.14), *nx* (3.6.14), *dx* (3.6.2), *dy* (3.6.2), *sx* (3.6.18), *sy* (3.6.18), *snx* (3.6.17), *sny* (3.6.17), *skx* (3.6.16) og *sky* (3.6.16). *lm ()* kaller kun *message ()* (3.8.30) av SIMBA Preprosessor spesifikke funksjoner. *mapfile* inneholder navnet på filen som skal leses.

3.8.25 loadall (file, maptype)

Dette er hovedsentralen for lasting av Kartfilsett (3.3). *loadall ()* kaller etter tur funksjonene *ld ()* (3.8.22), *lm ()* (3.8.24), *lv ()* (3.8.27), *li ()* (3.8.23), *message ()* (3.8.30) og *showmap ()* (3.8.48). Oppgavene utføres ved hjelp av følgende globale variable; *mainhandle* (3.6.10), *viewpoint* (3.6.22), og *mapname* (3.6.11). Kallet kommer enten fra *loadelements ()* (3.8.26) eller fra Kartdialogboksen (3.5.2.11). Parameteren *file* inneholder navnet på Kartfilsettet og parameteren *maptype* er en streng om angir visningsmodus for kartet.

3.8.26 loadelements ()

Callback for OKknappen i Gjenopprettdialogboksen (3.5.2.9). Igangsetter lasting av kart og tidligere lagrede plattformer fra fil. De global variable som nyttegjøres er *elements* (3.6.3), *utmx0* (3.6.19), *utmy0*, *nelem* (3.6.12) og *insertbefore* (3.6.6), og av SIMBA Preprosessor spesifikke funksjoner kalles i tur og orden *loadall ()* (3.8.25), *readelem ()* (3.8.39), *readcom ()* (3.8.38), *lwall ()* (3.8.28), *elplot ()* (3.8.21) og *message ()* (3.8.30).

3.8.27 lv (vegfile)

Så er vi tilbake til fillasting. Hvis det finnes en Vegetasjonsfil i Kartfilsettet (3.3.3), vil den bli lastet via denne funksjonen. Kallet kommer fra *loadall ()* (3.8.25), og selv kaller funksjonen bare *message ()* (3.8.30). Global variable ; *V* (3.6.21), *ny* (3.6.14) og *nx* (3.6.14). Parameteren *vegfile* inneholder navnet på filen som skal lastes.

3.8.28 lwall (vtemp, wallfile)

Her er funksjonen som laster Wallfilen fra Resultatfilsettet (3.2) over til den temporære wallfilen, hvis den eksisterer. Hvis den ikke eksisterer, forsøkes det å laste informasjon fra den temporære wallfilen direkte. Det er *wallfile* som refererer til Resultatfilsettets wallfil. *vtemp* inneholder navnet på den temporære arbeidsfilen. Til dette gjøres det bruk av den lokale funksjonen *isfile ()* for å finne ut om filen i *wallfile* eksisterer, og av *plotwall ()* (3.8.56) for å faktisk tegne strekene som representerer veggene. Kallet kommer fra *loadelements ()* (3.8.26).

3.8.28.1 isfile (fname)

Dette er en subrutine i filen lwall.m. Funksjonen sjekker om filen *fname* finnes.

3.8.29 makeveg ()

Dette er funksjonen som kalles som resultat av callback fra menyvalget Vegetation/Make i Hovedvinduet. Selv kaller den bare *message ()* (3.8.30). Det hentes inn to punkter på kartet fra brukeren. Etter hvert lagres veggen i den temporære wallfilen. Globale variable er *utmx0* (3.6.19), *utmy0* (3.6.19) og *nwall* (3.6.13), i tillegg til transportvariabelen *veg_height* som opprettes i callbacket til Vegetasjonsboksen.

3.8.30 message (str)

Ønsker man å dytte en tekst ut til Meldingsområdet i Hovedvinduet (3.5.1.1), kan denne funksjonen brukes. Den tar det som ligger i parameteren *str* og tilordner til feltet. Ingen kall, men den globale variabelen *mainhandle* (3.6.10) brukes. *message ()* kalles fra svært mange steder.

3.8.31 PerformAddElement ()

Denne funksjonen skulle nok egentlig hatt navnet PerformAddPlatform. Funksjonen kalles fra Applyknappen i Plattformdialogboksen når denne i sin tur ble kalt fra Plattformlistens Addknapp i Element Control. Den har det praktiske ansvaret for å dytte data for den nye plattformen inn i variabelen *elements* (3.6.3), samt å oppdatere status på en rekke knapper og lister i Element Control (3.5.1.2). Plattformen gjøres så aktiv i Plattformlisten ved kall til *changeplatform ()* (3.8.6). Nye plattformer legges alltid sist i både *elements* og Plattformlisten.

Funksjonen bruker de globale variablene *nelem* (3.6.12), *elements* (3.6.3) og *cmdhandle* (3.6.1). Fra denne funksjonene kalles også Tilleggsinfodialogboksen (3.5.2.12) for å hente ytterligere data, der det skilles mellom direkte- og indirekteskytende plattformer.

3.8.32 PerformAddObsdev ()

Kallet kommer fra OKknappen i Observasjonsmiddeldialogboksen (3.5.2.2) når denne blir kalt fra Observasjonsmiddellistens Addknapp. Det som utføres er at et nytt observasjonsmiddel legges til i plattformens OBS_DEVS liste i *elements* (3.6.3), samt at plattformens NOBS_DEVS inkrementeres. På slutten kalles *platforminfo ()* (3.8.31) for å oppdatere Element Control (3.5.1.2).

De globale variable som brukes er *nelem* (3.6.12), *elements* (3.6.3) og *cmdhandle* (3.6.1).

3.8.33 PerformAddWeapon ()

Denne funksjonen legger til et våpen i den aktive plattformens våpenliste. Fungerer helt likt *PerformAddObsdev ()* (3.8.32). Faktisk er det en kopi, der alt som har med observasjonsmidler å gjøre, er byttet ut med våpen.

3.8.34 PerformModElement ()

Denne funksjonen kalles fra OKknappen i Plattformdialogboksen når denne blir kalt fra Plattformlistens Modknapp i Element Control (3.5.1.2). Ansvarsområdet er stort sett bare å legge data fra dialogboksen inn i *elements*. Det kalles også på Tilleggsinfodialogboksen (3.5.2.12).

De vanlig globale variablene brukes også her; *nelem* (3.6.12), *elements* (3.6.3) og *cmdhandle* (3.6.1).

3.8.35 PerformModObsdev ()

Observasjonsmidlenes svar på *PerformModElement ()*. Det hentes data fra dialogboksen som overføres til *elements* før *platforminfo ()* kalles for å oppdatere Element Control (3.5.1.2).

Også her trengs de globale variablene *nelem* (3.6.12), *elements* (3.6.3) og *cmdhandle* (3.6.1).

3.8.36 PerformModWeapon ()

En blåkopi av *PerformModObsdev ()* (3.8.35) bare med endret fokus. Det er våpen det gjelder her.

3.8.37 platforminfo

Her er igjen en fil som mangler funksjonsdeklarasjon (linjen *function…).* Denne koden har da tilgang til alle variable fra den funksjonen den ble kalt fra, men den setter selv opp de globale variablene *elements* (3.6.3), *nelem* (3.6.12) og *cmdhandle* (3.6.1). Den eneste SIMBA Preprosessor spesifikke funksjonen som kalles fra *platforminfo ()*, er *UpdateCmd ()* (3.8.54). Ansvarsområdet omfatter oppdatering av Våpenlisten, Observasjonsmiddellisten og alle kommandolistene, samt styring av status for en del knapper. Kort sagt hele Element Control (3.5.1.2), unntagen Plattformlisten og Kommandolisten.

3.8.38 readcom (commfile)

Her leses kommandoer fra fil. Dette gjøres kun når brukeren laster et scenario fra filsett. Altså fra menyvalget Elements/Load. Kallet kommer fra *loadelements ()* (3.8.26). Globale variable som benyttes er *elements* (3.6.3), *utmx0* (3.6.19) og *utmy0* (3.6.19). Kall ut begrenses til *readwords ()* (3.8.40). *commfile* inneholder navnet på filen som skal leses.

3.8.39 readelem (elfile)

Her foregår den faktiske lesningen av Plattformfiler (3.2). *elfile* er filen som skal leses. Av globale variable brukes kun *elements* (3.6.3), men denne tømmes og opprettes på nytt. Kaller kun *readwords ()* (3.8.40).

3.8.40 readwords (string)

Vil man dele opp en tekst i enkelt ord er dette funksjonen å bruke. Resultatet er et array av ord. Teksten som skal deles ligger i *string*, og ordene returneres som resultat av funksjonen.

3.8.41 scom ()

Lagrer alle plattformenes kommandolister på fil. Kallet kommer fra *sfiles ()* (3.8.43), og av globale variable brukes kun *elements* (3.6.3) og *nelem* (3.6.12). Av kall er det bare *convert ()* (3.8.12) som er SIMBA Preprosessor spesifikt.

3.8.42 selem (efil)

Lagrer alle plattformer til filen *efil*. Til dette brukes de globale variablene *elements* (3.6.3) og *nelem* (3.6.12). Kalles fra *sfiles ()* (3.8.43).

3.8.43 sfiles (name, nscen)

Igangsetter lagring av aktivt scenario, med unntak av selve scenariofilen. Bruker ingen globale variable (unntatt *lhandle* som er en transportvariabel), og funksjonene *scom ()* (3.8.41), *selem ()* (3.8.42) og *swall ()* (3.8.52) kalles etter tur. *sfiles ()* kalles fra Lagredialogboksen (3.5.2.7).

3.8.44 showaddobsdevs ()

Klargjør og viser Observasjonsmiddeldialogboksen (3.5.2.2). Klargjøringen består stort sett av å legge tekst inn i IDfeltet og de to listeboksene. Data hentes fra filen *obsdevfile.mat* (3.7.1). Den globale variabelen *elements* (3.6.3) brukes for å finne IDnummeret til aktiv plattform. Kalles fra Addknappen til Observasjonsmiddellisten i Element Control (3.5.1.2).

3.8.45 showaddplatforms ()

Klargjør og viser frem Plattformdialogboksen (3.5.2.1). Klargjøringen består stort sett i å legge tekst inn i de to listeboksene. Data hentes fra filen *platformfile.mat* (3.7.2). Kalles fra Addknappen til Plattformlisten i Element Control (3.5.1.2).

3.8.46 showaddweapons ()

Klargjør og viser Våpendialogboksen (3.5.2.3). Klargjøringen består stort sett av å legge tekst inn i IDfeltet og de to listeboksene. Data hentes fra filen *weaponfile.mat* (3.7.4). Den globale variabelen *elements* (3.6.3) brukes for å finne IDnummeret til aktiv plattform. Kalles fra Addknappen til Våpenlisten i Element Control (3.5.1.2).

3.8.47 showelements ()

Åpner og initialiserer Element Control vinduet (3.5.1.2). Data hentes fra den globale *elements* (3.6.3), og *nelem* (3.6.12) brukes også. I tillegg initialiseres *cmdhandle* (3.6.1) i denne funksjonen. Kallet kommer fra menyvalget Platforms/Control i Hovedvinduet (3.5.1.1). Denne funksjonen fyller inn Plattformlisten selv, og setter en del knapper til start status, resten gjøres ved et kall til *platforminfo* (3.8.31).

3.8.48 showmap (type)

Dette er funksjonen som tegner kartet. Parameteren *type* angir hvordan det skal tegnes. Denne funksjonen bruker omtrent det som er av globale variable (3.6) men kall til andre funksjoner begrenser seg til de lokalt deklarerte subrutinene. Kallet kan komme fra en mengde steder.

3.8.48.1 plotveg (cx, cy, step)

Finnes i filen showmap.m. Tegner vegetasjonskart når kartmodusen er conveg.

3.8.48.2 fixaxes (fromx, tox, fromy, toy)

Finnes i filen showmap.m. Brukes til å sette tall på aksene.

3.8.49 showmodobsdevs ()

Gjør mye av det samme som *showaddobsdevs ()* (3.8.44), men i tillegg endres callbacket for dialogen. Kalles fra Observasjonsmiddellistens Modknapp i Element Control.

3.8.50 showmodplatform ()

Gjør mye av det samme som *showaddplatforms ()* (3.8.45), men i tillegg endres callbacket for dialogen, og ID settes i IDfeltet. Kalles fra Plattformlistens Modknapp i Element Control.

3.8.51 showmodweapon ()

Gjør mye det samme som *showaddweapons ()* (3.8.46), men i tillegg endres callbacket for dialogen. Kalles fra Våpenlistens Modknapp i Element Control.

3.8.52 swall (vtemp, fname)

Hvis det er lagt til, fjernet eller endret på vegger, må det skrives ny wallfil (3.2) fil. Dette utføres av denne funksjonen. Parameteren *vtemp* er navnet på den temporære wallfilen, og *fname* er navnet på den nye filen som skal lagres eller overskrives. Den globale variabelen *nwall* (3.6.13) brukes i funksjonen.

3.8.53 symbol (type)

Henter symbol for de forskjellige plattformene. For tiden har alle direkteskytende samme symbol; en fylt sirkel. Indirekteskytende plattformer har ingen symbol i denne forstand, ettersom de tegnes opp som omriss av nedslagsfelt. Selve symbolet (den fylte sirkelen) hentes fra filene *SYMBX.sym* og *SYMBY.sym* (3.7.3). Det er en subfunksjon i denne filen ved navn *symboltype ()*, og det er denne som gjør mesteparten av det lille arbeidet som skal gjøres.

3.8.54 UpdateCmd (ID, cnr)

Ansvaret som hviler på denne funksjonen består i å oppdatere Kommandolisten i Element Control (3.5.1.2). Den kalles fra to forskjellige steder; funksjonene *elcomand ()* (3.8.20) og *platforminfo ()* (3.8.31). Parametrene angir henholdsvis hvilken plattform som er aktiv og hvilken kommando som skal være aktiv etter at listen er oppdatert. De global variablene *elements* og *cmdhandle* benyttes, og det kalles ikke andre programspesifikke funksjoner.

3.8.55 viz (action)

Dette er funksjonen som mer eller mindre utgjør callbacket for alle valgene på Visability menyen. Parameteren *action* indikerer hvilket valg som ble gjort i menyen. *action* brukes så som argument i en switch/case konstruksjon. Av globale variable brukes *snx* (3.6.17), *dx* (3.6.2), *sny* (3.6.17), *dy* (3.6.2), *M* (3.6.8) og *isconplot* (3.6.7), og av funksjonskall er det kun *showmap*

() (3.8.48) og *message ()* (3.8.30) av SIMBA Preprosessor spesifikke funksjoner. I tillegg kalles Linjedialogboksen (3.5.2.8) eller Sektordialogboksen (3.5.2.13) for innhenting av tilleggsinformasjon.

3.8.55.1 CheckSecElement (x0, y0, alpha, h, dalpha, dh)

Denne funksjonen kaller en funksjon med navn *FreeLine ()* (3.8.55.2).Funksjonen kontrollerer om et ”grid-punkt” på kartet er synlig fra et annet. Det tegnes en gul firkant på kartet hvis den er synlig.

3.8.55.2 FreeLine (x1, y1, hobs, x2, y2, htarg)

Kontrollerer om det er fri sikt mellom punktet (*x1*,*y1*) og punktet (*x2*,*y2*). *hobs* er høyden over bakken for den som ser ut fra det første punktet, og *htarg* er tilsvarende for den som blir kikket mot i det andre. Således er det jo egentlig ikke fri sikt det er snakk om, men hvorvidt en fra høyde *hobs* over punktet (*x1*,*y1*) kan se noe med høyde *htarg* over punktet (*x2*,*y2*).

3.8.55.3 FreeThrSquare (ix, iy, sinn, sout, lx, ly, lz, x1, y1, z1)

Kontrollerer om det er fri sikt gjennom en grid-rute. Det er ruten med nedre venstre hjørne i punktet (ix,iy) som kontrolleres. Resten av parametrene er en gåte.

3.8.55.4 TerrainHeight (x, y)

Denne funksjonen finner høyden i et gitt punkt. Ettersom høydedataene er oppgitt for alle punkter der grid linjene krysser hverandre, mens det som regel er ruter som behandles, må det regnes ut en høyde for midten av ruten. En slags interpolering med andre ord.

3.8.56 wallplot (wallfile)

Her er funksjonen som faktisk tegner siktbegrensningsveggene på kartet under lasting av scenarier fra fil. Data hentes fra den temporære wallfilen som ble fylt av funksjonen *lwall ()* (3.8.28). Navnet på denne kommer inn via *wallfile*. Globale variable er *utmx0/utmy0* (3.6.19), og *mainhandle* (3.6.10). Kalles fra *lwall ()* (3.8.28).

3.8.57 whereami ()

Det hender at et program får behov for å sjekke hvor man er i filsystemet. Til dette benytter SIMBA Preprosessor denne funksjonen. Den returnerer rett og slett navnet på den filkatalogen du står i. Kalles fra *loadelements ()* (3.8.26), *lwall ()* (3.8.28), *makeveg ()* (3.8.29), *sfiles ()* (3.8.43) og *swall ()* (3.8.52).

4 DOKUMENTASJON AV KILDEKODE FOR KARTLOGG

4.1 Generelt

Programmet er skrevet i Microsoft Visual C++ (MSVC++), men ettersom programmet opprinnelig ble skrevet i strukturert C, er ikke all koden like gjennomført objektorientert. Det meste av strukturen skal være lagt om i versjon 4.1, men noe blir alltid hengende igjen. Det kan ved gjennomgang av dette kapitlet lønne seg å ha i bakhodet at KartLogg i all hovedsak er et fremvisningsprogram, og ikke en simulator. Følgelig er så godt som alle data fastsatt på forhånd og ikke regnet ut i kartlogg, og de er dermed faste for et scenario. De få data som regnes ut, er informasjon til programmet selv, og ikke noe som skal rapporteres. Dette medfører at verden sett fra programmets side kan forenkles en god del. De første avsnittene i dette kapitlet omhandler strukturen i programmet, mens de siste omhandler implementasjonen og de enkelte klassene.

4.2 Klasser

Artilleri	Data for Artillerienhet (Subklasse Av IndirectPlatform)
CaboutDlg	Dialogvindu for menyvalget ”Hjelp/Om Kartlogg”
CeventsToBeLoggedDlg	Dialogvindu for menyvalget ”Vindu/Hendelser”
CgetCoordsDlg	Dialogvindu for menyvalget ”Koordinater/Ta ut koordinater”
ClimitedInfoDialog	Dialogvindu for menyvalget ”Vindu/Kortinfo vindu”
CmapLogApp	Applikasjonsklasse. Entrypoint. Windows-overhead
CmapLogDlg	Programmets hovedvindu
CmapLogPrivate	Hjelpeklasse til CMapLogDlg (Hovedvindu)
CplatformDialog	Dialogvindu for menyvalget ”Vindu/Plattformer”
DirectPlatform	Superklasse for direkteskytende plattformer (Subklasse av Plattform)
FixedMapPosition	Liste/Data. Finnes i alle DirectPlatform’er. Inneholder waypoints
IndirectPlatform	Superklasse for indirekteskytende plattformer (Subklasse av Plattform)
InternalLogFile	Liste. Holder alle data fra logfilen under kjøring
LimitedInfoTransporter	Datatransport. Fra plattformer til KortinfoVindu, og tilbake
LogLine	Node/Data. Elementer i InternalLogFile
ObsDevList	Liste. Observasjonsmidler. Finnes i alle direkteskytende plattformer
ObsDevListNode	Node. Elementer i ObsDevList
PB	Data for PB enhet. (Subklasse av DirectPlatform)
Platform	Superklasse for Indirect- og DirectPlatform (legges i PlatformCollection)
PlatformCollection	Liste over plattformer
PlatformCollectionNode	Node. Elementer i PlatformCollection
Speed	Liste/Data. Fartsinformasjon for plattformer
Tank	Data for Tankenhet. (Subklasse av DirectPlatform)
View	Data for observasjonsmiddel. Legges i ObsDevList
Wall	Liste/Data. Siktbegrensningsinformasjon

Weapon	Data. Våpeninformasjon for plattformer
WeaponList	Liste over våpen tilhørende den enkelte plattform
WeaponListNode	Node. Elementer i WeaponList

4.2.1 Vinduer

Programmet er laget som en dialogvindu-applikasjon. Det er klassen til dette dialogvinduet (CMapLogDlg) som i bunn og grunn holder rede på alle dataene som brukes i programmet, og denne klassen finnes det kun én instans av. Vinduet har en hjelpeklasse (CMapLogPrivate). Innholdet i denne klassen er ikke privat for viduet i objektorientert forstand. Navnet henger igjen fra opprinnelig versjon der klassen eksisterte hovedsaklig for å redusere størrelsen på vindusklassen. Nå brukes den også til å forenkle andre klassers aksess til viktige datamedlemmer i hovedvinduet.

Ettersom KartLogg er et MS Windows program, må det finnes overhead. Dette faktum gjenspeiles i tilstedeværelsen til klassen CMapLogApp. Denne klassen har ingen egentlig funksjon, bortsett fra å gjøre det mulig å opprette dialogvinduets klasse. Når det gjelder de andre vinduene i programmet, så er de alle modale så nær som ett (CLimitedInfoDialog), og alle blir opprettet som barn av hovedvinduet. Alle vinduene har klasser med navn som ender med Dialog eller Dlg, og er således lette å identifisere i klasselisten. De vindusklassene som ennå ikke er nevnt ved navn i dette avsnittet, er CAboutDlg, CEventsToBeLoggedDlg, CGetCoordsDlg og CPlatformDialog.

4.2.2 Datalister

Mesteparten av dataene i programmet ligger lagret i en eller annen form for liste. De fleste store listene er bygget opp med en hovedklasse for selve listen og en elementklasse som tar seg av elementrelatert funksjonalitet (for eksempel holder listen sammen), men noen av de mindre listene er implementert som ”alt i ett”. Dette kan medføre at det blir vanskelig å se at det er en liste med elementer det er snakk om. Så får det bli opp til hver enkelt å vurdere om det er en fordel eller ikke. Alle listene vil bli beskrevet i dette dokumentet. De store listene implementert etter liste/node-prinsippet er InternalLogFile med sin LogLine (kombinert Node og Data element), ObsDevList med sin ObsDevListNode, PlatformCollection med tilhørende PlatformCollectionNode, og WeaponList med sin WeaponListNode. Videre har vi de listene som er dataobjekter og lister i ett, som omfatter klassene Speed, Wall, FixedMapPosition og View. I tillegg finnes en arvestruktur som inkluderer Platform, IndirectPlatform, DirectPlatform, Tank, PB og Artilleri. De eksisterer for å kunne legge ulike plattformer i samme liste. Altså for å få utnyttet en hensiktsmessig form for polymorfi.

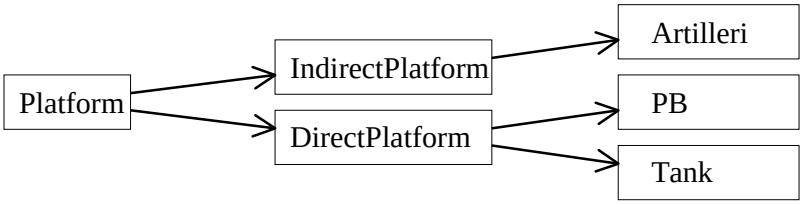
4.2.3 Data

Hva er et program uten data? Også i KartLogg må dataene oppbevares et sted. Klassene omtalt i avsnittene 4.2.1 og 4.2.2 gir struktur og brukergrensesnitt. De klassene som faktisk inneholder data er i all hovedsak Artilleri, PB og Tank, som representerer forskjellige platformtyper; LogLine, som holder én linje fra en loggfil, Speed (kombiliste), som holder fartsinformasjon, Wall (kombiliste), som holder siktbegrensningsinformasjon for kart, FixedMapPosition (kombiliste), som lagrer ”waypoint”-informasjon, View (kombiliste), som holder siktinformasjon, samt LimitedInfoTransporter, som er en klasse av den mer flyktige typen. Som navnet tilsier brukes LimitedInfoTransporter utelukkende til å transportere data.

4.3 Sammenhenger

4.3.1 Plattformene

Det finnes en klasse for hver type plattform som programmet er laget for å kunne håndtere. Ettersom det er av vesentlig betydning at ”en plattform er en plattform” enkelte steder i koden, har alle plattformene en felles superklasse. Ettersom nye plattformtyper implementeres, henges de bare på i strukturen. Foreløpig finnes det tre typer plattformer (trenger bare hovedtyper ettersom KartLogg i all hovedsak er et fremvisningsprogram), og de er organisert som følger:

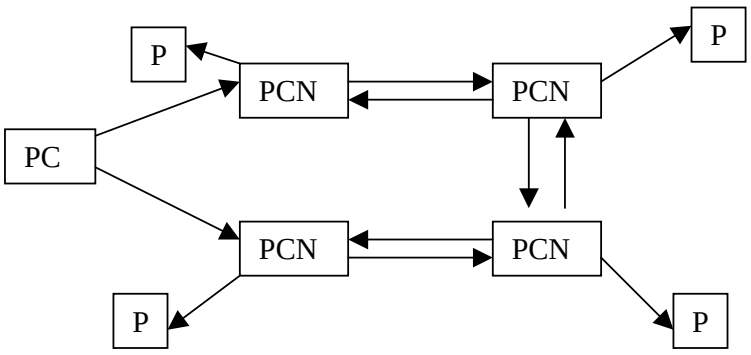


Figur 4.1: Klassestruktur for plattformene

For å opprette bindingene brukes vanlig arv, og for å nå spesifikk kode for den enkelte plattform, benyttes metoder som er deklarert ”pure virtual” i superklassen Platform. I tillegg til denne strukturen finnes det en liste som holder tak i pekere til alle plattformobjektene (avsnitt 4.3.2). I og med at alle tre plattformklassene har samme superklasse, kan de lagres som en peker til et objekt av typen Platform og dyttes inn i samme liste.

4.3.2 Platformlisten

Klassen PlatformCollection er implementert som vist i Figur 4.2 (eksempelet viser struktur med 4 plattformer). Når for eksempel alle plattformene skal tegnes opp på nytt (noe de må når skjermen oppdateres), kalles metoden DrawAllPlatforms i PlatformCollection, og bare den. Den sender så videre meldingen til alle enhetene i sin liste, i form av et kall til metoden DrawPlatform i klassen Platform. Denne metoden er deklarert som en ”pure virutal” metode og har kun som funksjon å videresende meldingen nedover i plattformhierarkiet (Figur 4.1). Enden på visen er at hver enkelt plattform, som jo selv vet hva den er, får beskjed om å tegne seg selv på nytt hvis det er nødvendig.

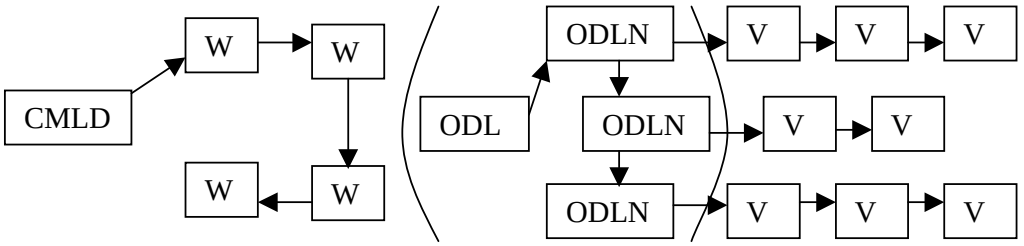


PC = PlatformCollection PCN = PlatformCollectionNode P = Platform

Figur 4.2: Listestruktur for plattformer

4.3.3 Kombilister

Denne gruppen omfatter, som tidligere nevnt, klassene Wall, Speed, FixedMapPosition og View. Disse listene er ikke lette å oppdage i koden, ettersom de ser ut som pekere til enkeltobjekter. Klassen View er litt spesiell fordi den er et dataobjekt for ObsDevList samtidig som den selv er en liste. Listen med Wall-elementer finnes det kun én av (i klassen CMapLogDlg: Wall* Walls), mens lister med Speed- og FixedMapPosition-elementer finnes i alle bevegelige plattformer. Strukturene blir som i Figur 4.3. Som man ser, finnes det ingen mellomledd mellom innehaver og data, og listeeieren har kun adressen til det siste elementet i listen (siste er her definert som det sist ankomne). Videre kan det bemerkes at listene er enkeltlenket. Alle metoder som vedrører listefunksjonene må derfor være rekursive på klassenivå.

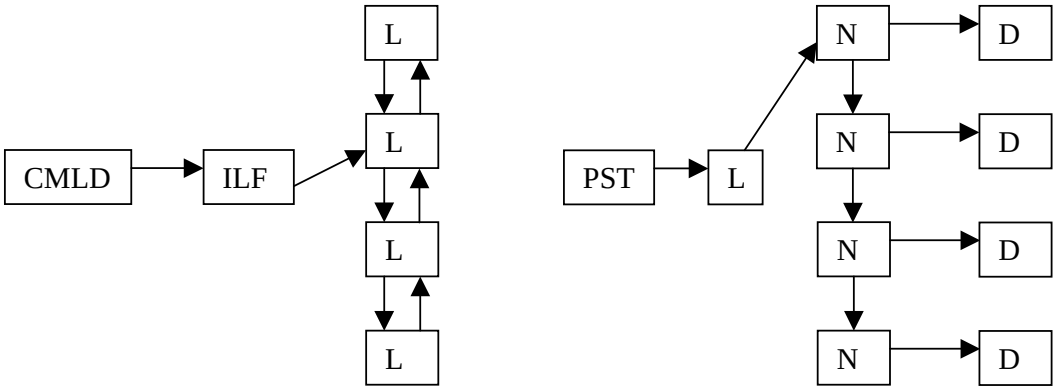


CMLD = CMapLogDlg ODL = ObsDevList ODLN = ODLNode
W = Wallobjekter V = View

Figur 4.3: Speed- og Wall-listene

Bruksmåten for de fire listene er noe forskjellig. Mens det for Speed- og FixedMapPosition-listen kun er siste elements data som er interessante, må data i alle elementene kunne nås i wall-listen. Wall-listen har derfor også metoder for datautnyttelse som er rekursive på klassenivå. Viewlisten har noe av det samme som wall-listen, men i tillegg kommer det at View inneholder flere parametre som kan settes separat. Derfor må tidligere instanster automatisk leses hvis gitt parameter i siste instans ikke er satt. Kun link til siste element er nødvendig, men en viss rekursivitet på klassenivå er nødvendig. Oppretting av nye instanser og fjerning av gamle, gjøres ved behov automatisk av listen selv.

4.3.4 Andre lister



CMLD = CMapLogDlg ILF = InternalLogFile PST = PlatformSubType L = Liste
LL = LogLine N = Node D = Data

Figur 4.4: Struktur for InternalLogFile, WeaponList og ObsDevList

Andre lister enn de nevnt i avsnitt 4.3.2 og 4.3.3 er InternalLogFile, ObsDevList og WeaponList. De to sistenevnte er enkeltlenket med peker kun til første element. De holder elementer som alle skal kunne aksesseres ved behov. InternalLogFile er litt mer komplisert, fordi dette er en liste som opprettes samlet (hele loggfilen leses i ett, og dermed opprettes alle elementene i listen samtidig), men ved bruk har man behov for å søke både frem og tilbake, samt å holde rede på hvor man er. Listen er derfor dobbeltlenket, med en flyttbar peker som eneste kontakt mellom listen og elementene. I tillegg kommer at elementene i InternalLogFile, objekter av typen LogLine, er kombinerte node- og dataelementer. Figur 4.4 viser strukturen for de to forskjellige listene. ObsDevList og WeaponList er helt likt implementert, derfor er det laget ett felles diagram for de to. For WeaponList er det klassene WeaponList, WeaponListNode og Weapon som er henholdsvis Liste, Node og Data, mens det for ObsDevListen er klassene ObsDevList, ObsDevNode og View som er det. Husk at View er en kombiliste i seg selv.

4.3.5 Vinduene

Alle vinduene i KartLogg opprettes som barn av hovedvinduet. Med unntak av CLimitedInfoDlg, er alle vinduene av ren dialogbokskarakter. Dette betyr at de overtar fokus fra programmets hovedvindu, og må avsluttes før fokus igjen flyttes til hovedvinduet (CMapLogDlg). Unntaket er som nevnt CLimitedInfoDlg. Dette vinduet blir stående og returnerer umiddelbart fokus til hovedvinduet. Sammen med hovedvinduet skal det dekke mesteparten av skjermen, men de to skal etter planen aldri overlappe hverandre.

4.4 Implementasjon

Resten av kapittel 4 er en gjenomgang av alle klassene. Deklarasjonene er gjengitt med de kommentarer som ligger i headerfilene, samt at ting av spesiell interesse eller vanskelighetsgrad er bemerket. For detaljer angående hvordan klassenes metoder er definert, (implementert) henvises det til kodefilene. Alle har navn av typen *klassenavn*.cpp. Utfyllende informasjon kan også i noen grad finnes i headerfilene, der dette er plassert utenfor selve klassedeklarasjonen.

4.5 Plattformer

I versjon 4.1 er det implementert Artilleri, PB og Tank. Klassen DirectPlatform kan ikke inneholde så mye mer eller mindre enn den gjør, men ettersom Artilleri er den eneste indirekte plattformtypen, er det problematisk å si sikkert hva som bør implementeres i IndirectPlatform fremfor i subklassen, men det er gjort et forsøk på inndeling. Endring av inndelingen er en enkel sak å utføre siden ved behov.

4.5.1 Klassen Platform

```
class Platform
{
public:
        //Constructorer
        Platform(int idNumber, int brigadeNumber, CString type);
        Platform(CString type);
        //Destructor
        virtual ~Platform();

        //Tegn platform på skjerm
        virtual void DrawPlatform (CClientDC* dcBmp, int logIndex, long time) = 0;
        //Utfør logglinje
        virtual void ExecuteLogLine (LogLine* logline) = 0;
        //Reversér utførelse av logglinje (ved kjøring baklengs)
        virtual void UndoLogLine (LogLine* logline) = 0;

        //Rapporter platformtypen (Artilleri, PB etc)
        CString PlatformType ();
        //Rapporter plattformens idnummenr (tre siffer)
        int IdNumber ();

        //Gir ut platformens displaystatus i form av flag
        virtual int DisplayState (int flags = -1) = 0;


protected:

        //Antall of halfkills platformen har påført motstanderne
        int HalfKills;
        //Skal platformen legges ut i LimitedInfoDlg vinduet?
        BOOL LimitedInfo;
        //Sette nå data til LimitedInfoDlg Endres
        BOOL UpdateLimitedInfo;
        //Bridagenummer
        int Brigade;
        //Plattformtype
        CString Type;
        //idNummer
        int Id;
};
```

Her er det ikke stort annet å bemerke enn at fire metoder er ”pure virtual”. Disse eksisterer fordi plattformene lagres i PlatformCollection som objekter av typen Platform, og følgelig kommer alle kall til plattformobjektene via klassen Platform. Av hensyn til programmererens korttidsminne brukes Pure Virtual for å garantere at disse metodene finnes i subklassene. Mangler de, får man kompileringsfeil. Legg også merke til at alle medlemmene er ”Protected”. Det betyr i praksis at det kun er objekter av subklasser som kan aksessere dem direkte. Alle andre må gå om metoder, og det defineres da metoder etter hva som er tillatt å

gjøre. Det er for eksempel ikke noe problem å få ut en plattforms ID-nummer, men å endre det er ikke mulig utenfra, fordi det ikke finnes noen metode for det. Hvordan får man satt det i utgangspunktet? Det gjøres ved opprettelse av subklassen, som så gir beskjed til superklassen. Som vi ser er denne klassen i hovedsak en paraplyklasse, som har til hensikt å gi alle plattformobjekter felles funksjonalitet uavhengig av hvilken type de er og hvordan de løser sine oppgaver.

4.5.2 Klassen DirectPlatform

```
class DirectPlatform : public Platform
{
public:
        //Constructor
        DirectPlatform(CString type);
        //Destructor
        virtual ~DirectPlatform();

        //Tegn platform til skjerm
        virtual void DrawPlatform (CClientDC* dcBmp, int logIndex, long time) = 0;
        //Utfør logglinje
        virtual void ExecuteLogLine (LogLine* logline) = 0;
        //Reversér utførelse av logglinje (ved kjøring baklengs)
        virtual void UndoLogLine (LogLine* logline) = 0;

        //Gir ut platformens displaystatus i form av flag
        virtual int DisplayState (int flags = -1) = 0;

protected:
        //Plattformen endrer retning
        void NewFixedPosition (long x, long y, long time, int heading);
        //Fjern siste faste punkt (ved baklengs kjøring)
        void RemoveFixedPosition ();

        //Rapporter siste passerte faste punkt for plattformen
        void MapPosition (long* x, long* y);
        //Rapporter når plattformen var ved siste faste punkt
        long TimeOfLastFixedPosition ();
        //Rapporter fartsretning (CCW fra øst)
        int CurrentHeading ();

        //Waypoint-liste
        FixedMapPosition* FixedPosition;
};
```

Å bemerke her er at medlemmet FixedPosition er en kombiliste (avsnitt 4.3.3) som lagrer alle de faste punktene som har vært definert for plattformen. Klassen arver fra klassen Platform, som vi kan se i øverst linje.

4.5.3 Klassen IndirectPlatform

```
class IndirectPlatform : public Platform
{
public:
        //Constructor
        IndirectPlatform(long, long, int, int, int, int, int, CString);
        IndirectPlatform(CString);
        InitIP (CString);
        //Destructor
```

```
        virtual ~IndirectPlatform();

        //Tegn plattform til skjerm
        virtual void DrawPlatform (CClientDC* dcBmp, int logIndex, long time) = 0;
        //Utfør logglinje
        virtual void ExecuteLogLine (LogLine* logline) = 0;
        //Reversér utførelse av logglinje (ved kjøring baklengs)
        virtual void UndoLogLine (LogLine* logline) = 0;
        //Gir ut plattformens displaystatus i form av flag
        virtual int DisplayState (int flags = -1) = 0;

protected:
        //Rapporter størrelsen på nedslagsfeltet i pixler
        BOOL TargetAreaScreenSize (int* width, int* height);
        //Rapporter nedslagsfeltets vinkel i grader (CCW fra øst)
        int TargetAreaAngleD ();
        //Rapporter nedslagsfeltets vinkel i radianer
        float TargetAreaAngleR ();
        //Rapporter nedslagsfeltets senter i skjerm koordinater
        BOOL TargetAreaScreen (int* x, int* y);

private:
        //sett nedslagsfeltets størrelse i kart koordinater
        void TargetAreaMapSize (int width, int height);
        //sett nedslagsfeltets senter i kart koordinater
        void TargetAreaMapCenter (long x, long y);

        //En haug med hjelpemetoder som forsåvidt godt kan fjernes (tror jeg)
        void TargetAreaMapHeight (int height);
        void TargetAreaMapWidth (int width);
        BOOL TargetAreaScreenWidth (int width);
        BOOL TargetAreaScreenHeight (int height);
        void TargetAreaAngle (int degrees);
        BOOL TargetAreaScreenCenter (int x, int y);
        BOOL TargetAreaScreenSize (int width, int height);
        int TargetAreaMapHeight ();
        int TargetAreaMapWidth ();
        int TargetAreaScreenWidth ();
        int TargetAreaScreenHeight ();
        void TargetAreaMap (long* x, long *y);

        //Nedslagsfeltetsvinkel i radianer
        float TargetAngleRadians;
        //Nedslagsfeltetsvinkel i Grader (CCW fra øst)
        int TargetAngleDegrees;

        //Målsenter på skjerm
        int TargetScreenY;
        int TargetScreenX;
        //Målstørrelse på skjerm
        int TargetScreenHeight;
        int TargetScreenWidth;

        //Målsenter på kart
        long TargetMapX;
        long TargetMapY;
        //Målstørrelse på kart
        int TargetMapHeight;
        int TargetMapWidth;
};
```

I denne klassen finnes atskillig mer funksjonalitet enn det som er tilfellet for DirectPlatform. Det vil nok bli en utjevning her, men det er ikke nødvendig. Hvis indirekteskytende plattformer har mer til felles enn direkteskytende, så er det riktig at mer utføres i superklassen for disse. Videre nevner jeg at det er lagt til grunn at artilleri (som foreløpig er den eneste plattform av denne typen) kun skyter mot ett mål (nedslagsfelt) og kun er aktiv én gang. Skal det skytes på samme sted flere ganger eller mot flere forskjellige mål, benyttes flere artillerienheter. Dette er en begrensning som må overholdes i SIMBA Preprosessor. Klassen arver fra klassen Platform som vi ser i øverste linje.

4.5.4 Klassen Artilleri

```
class Artilleri : public IndirectPlatform
{
public:
        //Constructor
        Artilleri(long, long, int, int, int, int, int, CString);
        Artilleri (CString, CArchive*, CString);
        //Destructor
        virtual ~Artilleri();

        //Tegner plattformen
        void DrawPlatform (CClientDC* dcBmp, int logIndex, long time);
        //Utfør en log linje
        void ExecuteLogLine (LogLine* logline);
        //Omgjør en utført loglinje
        void UndoLogLine (LogLine* logline);

        //Gir ut eller får inn plattformens displaystatus i form av flag
        int DisplayState (int flags = -1);

        //Hva med eget infovindu
        BOOL FullInfo;

private:
        //Rapporter antall avfyrte skudd
        int NumberOfShotsFired ();
        //Plattformen skyter nå
        void ShotFired (LogLine* logLine);
        //Plattformen lader
        void AddAmmo (int numberOfShots);

        //Sett start index
        BOOL StartIndex (int index);
        //Har plattformen started ennå?
        BOOL Started (int index);

        //Sett stopp index
        BOOL StopIndex (int index);
        //Har plattformen stoppet?
        BOOL Stoped (int index);

        //Er plattformen aktiv?
        BOOL Active(int index);

        //Startindex (Logglinjenummer)
        int Start;
        //Stopindex (Logglinjenummer)
```

```
        int Stop;

        //Antall skudd tildelt plattformen
        int TotalAmmo;
        //Antall skudd avfyrt
        int ShotsFired;
        //Artilleriets type (typisk våpenangivelse)
        CString SubType;

        //Var plattformen aktiv ved forrige kall til DrawPlatform
        //(Den trenger ikke slettes ellers)
        BOOL ActiveLast;

        int ShotsInAir;
        // Format: x, y, shottime, drawtime.
        int Shots [10][4];

        int DamageRadius;
};
```

Det er her det skjer noe for artilleriene. Metodene DrawPlatform, ExecuteLogLine og UndoLogLine er ikke lenger ”pure virtual”, og de må således ha en definisjon. At koden for tegning og logglinjetolkning kommer først her er helt logisk, ettersom det er først på dette nivået plattformen vet hva den er, og dermed hvordan dette skal utføres. Som vi ser, arver denne klassen fra klassen IndirectPlatform, som igjen arver fra klassen Platform. Vi ser også at denne klassen inneholder en del medlemmer som kunne ligget i superklassen. Hvis dette viser seg å være tilfelle også etter det kommer flere typer indirekteskytende plattformtyper, bør slik omplassering utføres.

En annen detalj som kan noteres, er at det i klassen Artilleri ikke er noen eksternt aksesserbare metoder utover de som fantes i superklassen. Dette er også logisk ved nærmere ettertanke, ettersom alle kall til Artilleri kommer via Platform og IndirectPlatform og således ville generert kompileringsfeil hvis de ikke fantes i superklassene. Det er slik MSVC++ sikrer at de sene bindingene (late bindings) er mulige. Legg også merke til at det finnes et array (Shots). Dette kan holde (huske) inntil 10 skudd, og disse skuddene slettes etter hvert som de ikke lenger skal tegnes.

4.5.5 Klassen Tank

```
class Tank : public DirectPlatform
{
public:
        //Constructor
        Tank(CString, CArchive* PlArc);
        //Destructor
        virtual ~Tank();

        //Tegner plattformen
        void DrawPlatform (CClientDC*, int, long);
        //Utfør en log linje
        void ExecuteLogLine (LogLine*);
        //Omgjør en utført loglinje
        void UndoLogLine (LogLine*);

        //Gir ut eller får inn plattformens displaystatus i form av flag
        int DisplayState (int flags = -1);
```

```
private:
        //Siktemidler
        ObsDevList ObsDevs;
        //Våpen
        int Weapons;
        //Er plattformen aktiv?
        BOOL Active (int);
        //Har plattformen startet
        BOOL Started (int);
        //Har plattformen avsluttet
        BOOL Escaped (int);

        //Sett startindex for plattformen (logglinjendex)
        BOOL StartIndex (int);
        //Sett stopindex for plattformen (logglinjendex)
        BOOL EscapeIndex (int);

        //Tiden siste gang plattformen ble flyttet på skjermen (logglinjeltid)
        long LastMoved;
        //Antall tildelte skudd
        int TotalAmmo;
        //Antall avfyrte skudd
        int ShotsFired;

        //Plattformens sluttindex (logglinjendex)
        int Escape;
        //Plattformens startindex (logglinjendex)
        int Start;

        //Liste over plattformens fartsendringer og når de fant sted
        Speed* CurrentSpeed;

        //Viser om plattformen har skudd underveis mot mål
        BOOL ShotInTheAir;

        //Markerer at skudd akkurat er avsluttet hvis TRUE
        BOOL ShotJustFinished;

        //Utregnet skjermposisjon basert på tid, fart, retning og
        //siste kjente faste posisjon
        int CurrentScreenX;
        int CurrentScreenY;

        //Skjermkoordinater for mål for aktivt skudd
        int ShotTargetX;
        int ShotTargetY;

        //Plattformens status. (MKilled && FKilled => TKilled)
        BOOL MKilled;
        BOOL FKilled;
        BOOL TKilled;

        //Stridsvognens type ( f.eks. T80)
        CString SubType;

        //Angir om plattformen var aktiv ved siste skjermoppdatering
        //Den trenger ikke slettes ellers
        BOOL ActiveLast;
```

```
        //Skal siktsektoren tegnes opp for denne plattformen?
        BOOL SightSectorOn;
        //Hva med eget infovindu
        BOOL FullInfo;
        //Skjul nummerlabel
        BOOL HideLabel;
};
```

Her gjelder det samme som for Artilleri hva eksternt aksesserbare metoder angår (se avsnitt 4.5.4). En annen likhet med Artilleriklassen er at det er her ting skjer for denne plattformtypen, av samme grunn. Vi ser at Tank arver fra DirectPlatform, som igjen arver fra Platform. Legg merke til medlemmet kalt CurrentSpeed. Dette er en kombiliste (avsnitt 4.3.3).

4.5.6 Klassen PB

```
class PB : public DirectPlatform
{
public:
        //Constructor
        PB(CString SubType, CArchive* PlArc);
        //Destructor
        virtual ~PB();

        //Tegner plattformen
        void DrawPlatform (CClientDC*, int, long);
        //Utfør en loglinje
        void ExecuteLogLine (LogLine*);
        //Omgjør en utført loglinje
        void UndoLogLine (LogLine*);

        //Gir ut eller får inn plattformens displaystatus i form av flag
        int DisplayState (int flags = -1);


private:
        //Siktemidler
        ObsDevList ObsDevs;
        //Våpen
        //int Weapons;
        WeaponList Weapons;
        //Er plattformen aktiv?
        BOOL Active (int);
        //Har plattformen startet
        BOOL Started (int);
        //Har plattformen avsluttet
        BOOL Escaped (int);

        //Sett startindex for plattformen (logglinjendex)
        BOOL StartIndex (int);
        //Sett stoptindex for plattformen (logglinjendex)
        BOOL EscapeIndex (int);

        //Tiden siste gang plattformen ble flyttet på skjermen (logglinjeltid)
        long LastMoved;
        //Antall tildelte skudd
        //int TotalAmmo;
        //Antall avfyrte skudd
        //int ShotsFired;
```

```
        //Plattformens sluttindex (logglinjendex)
        int Escape;
        //Plattformens startindex (logglinjendex)
        int Start;

        //Liste over plattformens fartsendringer og når de fant sted
        Speed* CurrentSpeed;

        //Viser om plattformen har skudd underveis mot mål
        BOOL ShotInTheAir;

        //Markerer at skudd akkurat er avsluttet hvis TRUE
        BOOL ShotJustFinished;

        //Utregnet skjermposisjon basert på tid, fart, retning og
        //siste kjente faste posisjon
        int CurrentScreenX;
        int CurrentScreenY;

        //Skjermkoordinater for mål for aktivt skudd
        int ShotTargetX;
        int ShotTargetY;

        //Plattformens status. (MKilled && FKilled => TKilled)
        BOOL MKilled;
        BOOL FKilled;
        BOOL TKilled;

        //Enhetens type ( f.eks. T80)
        CString SubType;

        //Angir om plattformen var aktiv ved siste skjermoppdatering
        //Den trenger ikke slettes ellers
        BOOL ActiveLast;

        //Skal siktsektoren tegnes opp for denne plattformen?
        BOOL SightSectorOn;
        //Hva med eget infovindu
        BOOL FullInfo;
        //Skjul nummerlabel
        BOOL HideLabel;
};
```

Også her er det verdt å legge merke til medlemmet CurrentSpeed. Dette er fremdeles en kombiliste slik som i klassen Tank. For øvrig gjelder de samme kommentarer som for Tank (avsnitt 4.5.5) også for resten av klassen

4.6 Plattformlisten

Nå er jo det meste røpet i avsnitt 4.3.2, så det gjenstår bare å se litt på klassedefinisjonene og bruksmåten. Vi husker at plattformlisten er en dobbeltlenket liste. Strukturen finnes i Figur 4.2 for referanse.

4.6.1 Klassen PlatformCollection

```
 class PlatformCollection
{
public:
```

```
        Platform* GetPlatformNumber (int);
        int NumberOfPlatforms ();
        //Constructor
        PlatformCollection();
        //Destructor
        virtual ~PlatformCollection();

        //Henter peker til en gitt plattform fra samlingen
        Platform* GetPlatform (int idNumber);

        //Videresender DrawPlatform til alle plattformer i samlingen
        void DrawAllPlatforms(CClientDC* dcBmp, int logIndex, long time);

        //Legger den nye plattformen til samlingen
        void AddPlatform (Platform* platform);

        //Flytter en plattform sist i listen slik at den blir
        //tegnet øverst på skjermen
        BOOL LiftPlatform (int pId);

        //Tømmer samlingen og frigjør plassen som var okkupert
        void Empty();

private:
        //Sjekker om en plattform finnes
        //BOOL IsPlatform (int idNumber);

        //Finner noden til en plattform med gitt IdNummer
        PlatformCollectionNode* FindPlatformNode (int idNumber);
        //Antall elementer i samlingen
        int NumberOfNodes;

        //Link til selve samlingen
        PlatformCollectionNode* Head;
        PlatformCollectionNode* Tail;
};
```

Dette er inngangsporten til plattformsamlingen. Koden over er vel nokså selvforklarende med de kommentarene som ligger der. Som man ser, refereres plattformene stort sett alltid ved id-nummeret fra plattformfilen. Plattformenes rekkefølge i listen er vesentlig. De får beskjed om å tegne seg fra ene enden av listen. Dette utnyttes av metoden LiftPlatform, som rett og slett flytter plattformer internt i listen for å endre tegnerekkefølgen.

4.6.2 Klassen PlatformCollectionNode

```
class PlatformCollectionNode
{
public:
        //Constructor
        PlatformCollectionNode(Platform*);

        //Legg til ny node
        PlatformCollectionNode* Insert (PlatformCollectionNode* newNode);

        //Slett noden som mottar meldingen, og frigi dens data
        //returnerer peker til neste node i samlingen
        PlatformCollectionNode* Delete ();

        //Lever ut peker til nodens data
```

```
        Platform* GetPlatform (void);

        //Lever ut peker til forrige element i samlingen
        PlatformCollectionNode* PreviousNode ();

        //Lever ut peker til neste element i samlingen
        PlatformCollectionNode* NextNode ();

        //flytt noden sist i listen (Den blir da tegnet sist)
        void Float ();

        //Operator overload
        //Tester idnummer hos nodens data mot parameter
        BOOL operator== (int idNumber);

private:
        //Sett peker til neste node
        void SetNext(PlatformCollectionNode* newNext);
        //Sett peker til forrige node
        void SetPrevious(PlatformCollectionNode* newPrevious);

        //Destructor
        virtual ~PlatformCollectionNode();

        //Peker til nodes data (en Plattform)
        Platform* Data;
        //Peker til neste node
        PlatformCollectionNode* Next;
        //Peker til forrige node
        PlatformCollectionNode* Previous;
};
```

I klassedefinisjonen legger vi merke til to ting. Det første er at destructoren er deklarert i ”private” delen. Dette er fordi all sletting av nodeobjekter skal tvinges via metoden Delete. Grunnen er at det må utføres sammenkobling av den resterende listen før sletting, og det går ikke via den vanlige destructoren. Det andre som legges merke til er at det finnes en ”overloaded operator”. Normalt ville utrykk av typen ”node* == int” testet hvorvidt de to argumentene inneholdt samme tallverdi, altså adressen til samme nodeobjekt. Den nye operatoren fjerner denne tolkningen og erstatter den med en test på hvorvidt nodens datamedlem sitt ID-nummer stemmer overenes med nummeret gitt som parameter nummer to. Dette forenkler koden for gjenkjenning av spesifikke plattformobjekter i listen. Formen blir nå ”node* == int”, og i praksis er det ”node*->data->IdNumber == idNumber” som blir testet.

En annen ting som er litt spesielt, er at man ved eksekusjon av logglinjer sender linjen til listen, som så finner rett plattform og sender linjen videre, i motsetning til den noe mer strukturmessig rene løsningen der man ber om å få ut aktuell plattform fra listen og så sender linjen til plattformen fra det sted der linjen i utgangspunktet ble tolket. Dette er gjort fordi det virket enklere. Nå trenger nemlig ikke logglinjetolken å vite hva en plattform er.

4.7 InternalLogFile

Når det angis hvilken loggfil som skal brukes, leses denne inn i minnet i sin helhet. Samtidig benyttes anledningen til å lagre dataene på et mer hensiktsmessig format. Hver linje fra loggfilen lagres som ett element i listen InternalLogFile. Listen er enkel i det det kun finnes ett dataelement for hver node.

4.8 Vinduene

Det er begrenset hvor interessant vinduenes kode er. Her gjengis de av headerfilene det synes å være noe i, som er verdt å nevne. For de andre refereres det til filnavet på den aktuelle filen.

4.8.1 Hovedvinduet

```
class CMapLogDlg : public CDialog
{
// Construction
public:
        //antall loglinjer i gjeldende loggfil?
        int         NrOfEvents;

        //Kartstørrelse
        long XMap1, XMap2;
        long YMap1, YMap2;
        long XScreen1, XScreen2;
        long YScreen1, YScreen2;

        //kontrollvariable
        BOOL MapFileLoaded;
        BOOL ReadyToGetCoord;
        BOOL PointNr1;
        BOOL GottenCoords;
        BOOL ShowNow;
        BOOL LogFileLoaded;
        BOOL ScenarioFileLoaded;
        BOOL DeleteBeforeDraw;

        //Navn på input-filer
        CString MapFileName;
        CString LogFileName;
        CString ScenarioFileName;
        CString PlatformFileName;
        CString CoordFileName;
        CString CommandFileName;

        //Plattformlisten
        PlatformCollection PlatformList;
        //Logfil kopien
        InternalLogFile LogList;

        //Statisk vindu med av/på funksjon
        CLimitedInfoDialog* LimitedInfoDlg;

        //Data som regulerer lesing fra loggfil
        CEventsToBeLoggedDlg EventsToBeLogged;

        //Hjelpeklasse
        static CMapLogPrivate* MapLogPrivate;

        //Filbehandling
        BOOL OpenFiles(CString*, char*, CString, CString);
        void SaveFile(CString *FileNameP, char *Filter,
                                                CString InitialDir, CString Title,
                                                CString defaultExt);

        //Tegn opp kartet på skjermen
        void PaintMap();
```

```
        // standard constructor
        CMapLogDlg(CWnd* pParent = NULL);

        //Link til kartbildet
        PDIB m_pdibPicture;

// Implementation
protected:
        PDIB DibOpenFile(LPSTR szFile);
        PDIB DibReadBitmapInfo(HFILE fh);


private:
        //Metoder for lesing av diverse filer
        void LoadWalls ();
        BOOL LoadLogfile ();
        BOOL LoadScenario ();
        BOOL LoadCoord ();
        BOOL LoadMap ();

        //Wall-liste
        Wall* Walls;

        //Enhet- og type-konvertering får retningsdata
        double degIToRadD (int degrees);
};
```

Man ser fort at det er dette vinduet som er programmets kjerne. I denne klassen finner man både InternalLogFile-listen, Platform-listen og Wall-listen direkte i klassedefinisjonen. Det vil si, de opprettes ikke her, men det er herfra de kan aksesseres. De opprettes og fylles, og slettes hvis nødvendig, i metodene merket ”Metoder for lesing av diverse filer”. For å lette gjennomgangen er all ”messaging”-informasjon fjernet fra klassedeklarasjonen. Dette er spesifikke windowsting som ikke egentlig har noe med implementasjonen å gjøre. Vi ser at MapLogPrivate deklareres som ”static”. Dette gjøres for lettere å få tilgang til data fra objektet. Ved å gjøre det på denne måten, trenger man ikke ha peker til vinduet eller vinduets klasse for å aksessere MapLogPrivat. Man trenger kun vite navnet på klassen, medlemmet og metoden.

4.8.2 Klassen CMapLogPrivate

```
class CMapLogPrivate
{
public:
        //Setter styreflagg
        void SetValues (CEventsToBeLoggedDlg*, CLimitedInfoDialog*);

        //kartstørrelse
        long XMap1, XMap2;
        long YMap1, YMap2;
        long XScreen1, XScreen2;
        long YScreen1, YScreen2;

        //Styreflagg for skjermvisning
        BOOL ShowSectors;
        BOOL ShowLimitedInfo;

        //Peker til vinduer
```

```
        CEventsToBeLoggedDlg *EventsToBeLogged;
        CLimitedInfoDialog* LimitedInfoDlg;

        //Constructor
        CMapLogPrivate();

        //Hent og konverter koordinater
        void GetCoordData(long[8]);
        void ScreenToMapCoord(long*, long*, long, long);
        void MapToScreenCoord(long*, long*, long, long);
        long MapToScreenDistance (long MapDistance);

        // Åpner logfilen og sender enkeltlinjene til LogList
        BOOL MakeEventList(CString, InternalLogFile*, CSpinButtonCtrl*, int*);

        //Lager en linket liste over plattformer utifra startfila
        //platforms.xx som står i scenariofila.
        BOOL MakePlatformList(CString, CString, PlatformCollection*);

        //Sjekker om strengen/hendelsen er av en type som skal logges.
        BOOL CheckEventType(CString);

        //Metode for å hente filnavnene til loggfilen og plattformfilen ut fra det som står i scenariofilen
        //Koden går ut fra at plattformfilen ligger i samme katalog (data) som scenariofilen.
        //Metoden finner også kommandofilen.
        BOOL ReadScenarioFile(CString, CString*, CString*);

        //Henter data fra kommando filen
        CString GetFromCommandFile (int id, CString lineType, CString CommandFile);
}
;
```

Som man ser er det en del av medlemmene i CMapLogDlg som gjentas i denne klassen. Det er strengt tatt ikke nødvendig å ha det slik, men skal dette rettes opp, bør de slettes i CMapLogDlg, ettersom CMapLogPrivate er instansiert som ”static” der, og dens medlemmer og metoder dermed er tilgjengelige fra hvor som helst i koden.

4.8.3 Klassen CLimitedInfoDialog

```
class CLimitedInfoDialog : public CDialog
{
public:
        // Constructor
        CLimitedInfoDialog(CWnd* pParent = NULL);

        //Avslutt visning av en plattform
        void RemovePlatform (int);
        //Motta data for en plattform
        void ReceiveData (LimitedInfoTransporter*);

private:
        //Link til hovedvinduet
        CWnd* MainWindow;
        //Listeboksen i vinduet
        CListBox* List;
        //VInduets bredde
        int ListBoxCanvasWidth;
};
```

Dette er vinduet som benyttes til å vise såkalt kortinfo (menyvalget ”Vindu/KortInfo vindu”). Dette er det eneste vinduet som ikke må stenges før videre interaksjon med hovedvinduet kan foretas. Dette vinduet er altså ikke ”modalt”, og følgelig må det opprettes på en litt annen måte enn resten av vinduene. Det opprettes faktisk i konstruktoren til CMapLogDlg (hovedvinduet), og eksisterer dermed så lenge programmet kjører. At et vindu eksisterer er imidlertid ikke det samme som at brukeren kan se det, og det slås altså av og på via menyen. Fordi det ved visuell opprettelse må sette fokus tilbake til hovedvinduet, trenger det link til dette. Det eneste visuelle innholdet i vinduet er en listeboks, som deklareres spesifikt i klassen. Denne listeboksen tar seg av sortering av linjene automatisk. For interesserte er det i metoden CMapLogDlg::OnMenuLimitedinfo at vinduet visuelt slås av og på. Data mates inn via metoden ReceiveData av hver enkelt plattform, når plattformene selv mener det er nødvendig. Dette er således et passivt vindu i den forstand at det ikke selv aktivt henter data, men kun er et display.

4.8.4 Andre vinduer

I tillegg til de tidligere nevnte, finnes det fire vinduer som har egne klasser. Alle er av typen Modal og relativt lite spennende. Filene heter det samme som klassene, med unntak av CAboutDlg, som er deklarert i filen CMapLogDlg sammen med hovedvinduet.

CAboutDlg er den klassiske Aboutboksen i MS Windows. Den opprettes automatisk av MSVC++ når et nytt prosjekt startes. Det kan derfor være litt kludrete å finne ut hvor innholdet hentes fra. Filen MapLog.rc er svaret, men den må åpnes som en tekstfil.

CEventsToBeLoggedDlg gir mulighet til å endre kjøringsmodus. Utover dette fungerer vinduet sånn at programmet ikke laster logglinjer i kategorier som ikke er valgt i dette vinduet. Vinduet åpnes fra menyen ”Vis/Hendelser”.

CGetCoordsDlg brukes ved manuell inntasting av kartkoordinater. Den åpnes to ganger på rad, for inntasting av to punkter, og dermed må det en styrevariabel til for å vite om det er første eller andre punkt som leses. Hele denne styremekanismen er implementert i CMapLogDlg::OnLButtonDown, og vinduet selv trenger derfor svært lite intelligens. Vinduet trigges først av menyvalget ”Koordinater/Ta ut koordinater” etterfulgt av venstreklikk på kartet, og igjen av et nytt venstreklikk på kartet.

CPlatformDialog er det siste vinduet, og brukes for å spesifisere hva som skal vises for den enkelte plattform. Dette er på mange måter det mest spennende av de modale vinduene, med alle sine hakebokser. Det er implementert slik at valgene bekreftes hver gang ny plattform velges eller knappen ”Lukk vindu” trykkes, men på skjermen gjøres det ikke endringer før skjermen oppdateres av andre grunner. Dette vinduet oppdaterer altså ikke skjermen selv. Valget ”Fullstendig informasjon” er ikke implementert.

I tillegg til de ovennevnte programspesifikke vinduene vil brukeren komme i kontakt med de to MSWindows-standardvinduene for å åpne og lagre filer. Disse brukes slik standardvinduer alltid har blitt brukt i MSWindows, ved å overføre en haug med kryptiske strenger i en ”struct”.

4.9 Listestrukturer

Som beskrevet i avsnitt 4.2.2, og i avsnittene 4.3.2, 4.3.3 og 4.3.4, holdes programmets data i forskjellige lenkede lister. Slike lenkede lister medfører alltid en risiko for minnelekkasjer.

Det er ingen kjente lekkasjer i programmet slik det foreligger, men ved eventuelle fremtidige utvidelser av KartLogg bør man være klar over at MSVC++, versjon 4, ikke rapporterer slike lekkasjer. Versjon 6 derimot, varsler programmereren etter kjøring i debugmodus (versjon 5 er ikke testet).

Som for vinduene (avsnitt 4.8) er det ikke alle listene som representerer noe nytt, spesielt, eller særlig interessant. Samme fremgangsmåte følges derfor for dokumentasjon av disse vinduene.

4.9.1 Observasjonsmiddellisten

Listen består av klassene ObsDevList (liste), ObsdevListNode (node) og View (data), og er ganske rett frem helt til man begynner å se på dataklassen. Men det første først. Det opprettes en liste for hver plattform som har observasjonsmidler, og en node for hvert observasjonsmiddel. Det er ingen begrensning på antall observasjonsmidler som kan ligge i denne listen, men vær opmerksom på at det i forbindelse med innlesning fra platformfilen (også kalt elementfil) ligger en begrensning på tre observasjonsmidler til hver plattform.

```
class View
{
public:
        //Er alle parametre satt minst en gang?
        BOOL Ready ();
        //slett hele listen
        void DestroyAllViews ();
        //Constructors
        View();
        View (View*);

        // Hent opplysninger fra View kjeden
        int ViewSector ();
        int ViewDirection (int, int);
        long ViewRange ();

        // Sett eller slett data i View kjeden
        View* ViewSector (int);
        View* ViewRange (long);
        View* ViewDirection (int direction = -1);
        View* ViewPoint(int x = -1, int y = -1);

        //Slett en instans
        View* Delete ();

private:
        //Initialiser instansen med ”umulige verdier”
        void InitialiseView ();
        //Destructor
        virtual ~View();
        //Regne ut retning ved punktfølging
        void CalculateViewDirection (int, int);

        //Data medlemmer
        long Range;
        int Sector;
        int Direction;
        int Y;
        int X;
```

```
        BOOL Point;

        //Link til forrige instans
        View* PreviousView;
};
```

Så var det dataene. Det som må lagres for hvert observasjonsmiddel, er siktsektorens lengde, retning og vinkelbredde, samt hvorvidt plattformen ser på et bestemt punkt eller ikke. Hvis plattformen ser på et punkt, må retningen oppdateres hver gang plattformen flytter seg. Alle disse verdiene kan endres individuelt, men det gjøres svært sjelden i praksis. Følgelig fant vi det hensiktsmessig å lage en struktur som ikke kopierer gamle data for de enhetene som ikke endres. Denne listen er derfor implementert slik at det ved setting kontrolleres hvorvidt de aktuelle dataposter er satt fra før. Hvis de er det, opprettes en ny instans av View, der kun aktuell datapost settes. Er ikke dataelementet satt, settes det i (siste) eksisterende instans.

Når man så skal lese de lagrede data for et observasjonsmiddel, er det de sist satte verdiene som er interessante, og *view* kontrollerer da om siste instans har elementet satt. Hvis ikke, så sjekker den elementet i den forrige instansen. I tillegg kontrolleres det at alle elementene har vært satt minst én gang før lesing tillates. Grunnen til at gamle data trengs å lagres er at det kan kjøres baklengs i loggfilen. Når dette gjøres, slettes aktuelt element, og hele instansen slettes hvis slettingen av elementet resulterer i en tom instans. På denne måten trenger ikke loggfilen gjennomsøkes for å finne forrige verdier for de aktuelle parametre. Alle parametre settes ved at logglinjen sendes direkte til listens ExecCom eller UndoCom metoder på øverste nivå, altså i ObsDevList.

4.9.2 Kombilistene

Som det står i avsnitt 4.3.3, omfatter disse klassene FixedMapPosition, Wall, Speed og View. View er omtalt grundig i avsnitt 4.9.1, og resten av dette avsnittet handler følgelig om de tre andre. Alle disse er implementert slik at de instansieres med alle data som parametre. Ettersom eieren bare har én lenke til listen, må det sendes inn link til det hittil siste elementet i tillegg. Linken til listen tilordnes deretter det siste elementet man akkurat instansierte. Kallet kan typisk se slik ut:

```
Speeds  = new Speed (<farten>, Speeds);
```

Her ser man at variabelen Speeds brukes to ganger på samme linje. Først som parameter og så som tilordningsobjekt. Dette gir den nye instansen mulighet til å lagre den gamle lenken, samtidig som Speeds oppdateres med den nye.

4.9.3 Weaponlisten

Weaponlisten er alt i alt den enkleste listen i programmet. Den er en implementasjon i tre ledd og består av listehode, listenode og dataelement implementert i WeaponList-, WeaponsListNode- og Weaponklassene. Denne listen har dessuten bare ett dataelement for hver listenode, og fungerer bare som en lagringsplass for navnene på plattformens våpen. Det er her det blir viktig å huske at vi ikke skal simulere. Følgelig er våpnenes parametre, som kraft og rekkevidde, ikke interessante. Samme begrensning gjelder her som for ObsDev-listen (avsnitt 4.9.1); Listen kan ta så mange elementer man vil, men ved lesing fra fil tolereres bare tre.

5 DOKUMENTASJON AV KILDEKODE FOR POSTPROSESSOR

5.1 Generelt

SIMBA Postprosessor er et verktøy for utregning av resultater fra simuleringer gjort i SIMBA Simulator. SIMBA Simulator produserer loggfiler fra simuleringene, og disse mates så til SIMBA Postprosessor, som regner ut diverse summer, gjennomsnitt og standardavvik. SIMBA Postprosessor kan også brukes som hjelpeverktøy til SIMBA Preprosessor, i den forstand at det kan være enklere å kontrollere at input til SIMBA Simulator er riktig ved å se på output. Brukes SIMBA Postprosessor i en slik sammenheng, anbefales det å også benytte postprosesseringsverktøyet SIMBA Kartlogg. Det Postprosessor kan bidra med er vel stort sett å bryte ned logg-filen og fordele den på de enkelte plattformer.

Det er viktig å ha i bakhodet at SIMBA Postprosessor ikke er et simuleringsverktøy. Derfor kan verden forenkles sett fra programmets kildekode. Det blir for eksempel ikke nødvendig å vite hvilken type ammunisjon som benyttes, og hvilken skade den er kapabel til å gjøre på forskjellige mål. All slik informasjon tas hensyn til i SIMBA Simulator. SIMBA Postprosessor er et rent regneverktøy, og får alle resultatene av skudd servert med resultat i klartekst via loggfilene.

5.2 Hovedstrukturer

I SIMBA Postprosessor finnes det to hovedstrukturer for datalagring. Den ene holder alle data fra en logg-fil, og den andre er en liste over plattformer i en simulering, med underliggende liste med alle logg-linjer som gjelder den enkelte plattformen. Begge disse gjelder enkelt-filer. Ettersom en simulering kan bestå av gjentatte kjøringer og dermed produsere mange logg-filer, trenger vi et sted å akkumulere data. Dette er imidlertid ikke nødvendig før operatøren ber om en resultatliste eller en resultatmatrise. Derav anses dette ikke som hovedstrukturer i PostProsessor.

5.2.1 Intern kopi av loggfilen

Klassen InternalLogFile holder til enhver tid en strukturert kopi av loggfilen som ligger til grunn for det som vises i hovedvinduet. Hver linje fra filen er representert ved en instans av klassen LogLine, og alle disse instansene er lenket sammen og kontrolleres av InternalLogFile. Denne strukturen fungerer som en dobbeltlenket pekerliste.

5.2.2 Plattformene

Hver plattform i en simulering får i SIMBA Postprosessor en instans av typen Platform. Denne instansen inneholder en observasjonsmiddelliste (ObsDevs), en våpenliste (Weapons), en skuddliste (Shots) og en hendelsesliste (PlatformEvents). Alle disse listene inneholder utelukkende informasjon om den plattformen instansen representerer. For visning i hovedvinduet er kun data fra én fil aktuelt, og følgelig er det ikke noe problem å bruke denne strukturen før dataene dyttes inn i de ulike GUI-elementene og lagres der. For resultatlisten og resultatmatrisen blir det noe mer komplisert, ettersom det her kan være snakk om mange filer, veldig mange. Dette er løst ved at all statistikk regnes ut for hver fil og lagres i matriser

tilpasset behovet. Når neste fil er prosessert, adderes resultatene inn i matrisene. Platformlisten blir således redusert til et transittlager for informasjon.

5.3 Vinduer

PostProsessor består av totalt 4 vinduer, der 3 er modale pop-up vinduer som alle har hovedvinduet som nærmeste foresatte, og det 4. er hovedvinduet selv. Alt arbeidet som utføres for å skaffe data til de tre modale vinduene, trigges av at det aktuelle vinduet genereres. Altå er metodene onInitDialog () gode plasser å begynne hvis man vil se detaljert på hvordan koden virker.

5.4 Klasser

Klassene som inngår i Postprosessor er :

- CAboutDlg : Det klassiske MSWindows-About-vinduet. Trigges fra menyvalget ”Help/Om programmet”.
- CEndResultDlg : Viser simuleringsresultater i form av total score for hver side. Trigges fra knappen ”Resultatliste” i hovedvinduet.
- CLookAtDlg : Vindu for visning av filer i tekstmodus. Trigges av ”Se på fil knappen” i hovedvinduet.
- CLookAtFileDlg : Vindu for inspeksjon av filer. Her kan man ta en titt på aktuelle senario-, element- og loggfiler.
- CMakeList : Tolker filer og mater forskjellige lister.
- CPostProcessorApp : Typisk MSWindows-overhead. Dette er i tillegg programmets Entry-point, altså der koden starter.
- CPostProcessorDlg : Hovedvinduet
- CResultMatrixDlg : Viser simuleringresultater i form av en matrise med blå og oransje plattformtyper langs aksene. Trigges av knappen ”Resultatmatrise” i hovedvinduet.
- CStatistics : Regner ut resultatene før de overføres til det aktuelle vindu.
- InternalLogFile : Holder en liste med objekter av typen LogLine. Finnes både som frittstående med hele loggfilen, og som plattformavhengig med kun utdrag.
- LogLine : Formatert versjon av en linje fra loggfilen. Lagres i InternalLogFile, hvor nå den måtte befinne seg.
- ObsDevList : Liste over siktemidler. Holder en kjede med objekter av typen View.

- Platform : Representasjon av en plattform. Alle plattformer er like sett fra PostProsessor sitt synspunkt.

- PlatformCollection : Plattformliste. Tredelingen av denne listen henger igjen fra implementasjonen av SIMBA KartLogg, der koden er hentet fra. Tar elementer av typen PlatformCollectionNode.

- PlatformCollectionNode : Pekeradministrator for plattformlisten. Holder en plattform hver, samt peker til forrige og neste PlatformCollectionNode.

- Shot : Hvert avfyrt skudd representeres av en Shot-instans. Det inkluderer også skuddets effekt. Listes i hovedvinduet

- ShotList : Skuddliste. Samler alle skudd fra en plattform. Holder kjede av Shot-instanser.

- View : Et siktemiddel. All informasjon som er tilgjengelig for et siktemiddel lagres i en instans av denne typen. Går som medlem i en ObsDevList. Listes i hovedvinduet.

- Weapon : Et våpen, eller en ildgivningsenhet om man vil. Det som lagres er alt som er tilgjengelig av informasjon om dette våpenet. Disse objektene nyttes først og fremst til visning i hovedvinduet.

- WeaponList: Holder kjede av Weapon-instanser. Alle våpen på en plattform samles i en våpenliste.

APPENDIKS

A BESKRIVELSE AV LOGGFILER FRA SIMBA

A.1 Formål

Formålet med dette kapitlet er å beskrive loggfilene som genereres av SIMBA. Loggfilene er tekstfiler som beskriver hendelsesforløpet for en enkelt simulering, der én linje beskriver én enkelt hendelse. Loggfilene brukes som input til SIMBA postprosessor (SP) og SIMBA kartlogg (KL). Endringer i SIMBA som fører til endringer i loggfilene, vil derfor føre til behov for endringer også i disse programmene.

A.2 Hendelsestyper

For hver simulering kan man velge hvilke hendelsestyper som skal logges. SIMBA bruker følgende hendelsestyper:

- ***Basis (B)***. Basislinjene er de linjene som **må** være med for at SP og KL skal fungere. En stor del av linjene kommer i denne kategorien, som ikke kan utelates
- ***Deteksjon (D)***. Inneholder informasjon som angir om plattformene kan oppdage hverandre, og om de faktisk gjør det.
- ***Missilbevegelser (M)***. Forteller hvor langt fra målet et bestemt missil er på et gitt tidspunkt.
- ***Ildprosedyre (F)***. Inneholder informasjon om lading og sikting.
- ***Siktelinjer (V)***. Beskriver størrelse og retning til plattformenes siktsektorer. Ved å sløyfe denne kategorien, mister man noe av funksjonaliteten til KL.
- ***Resten (#)***. Noen få linjer passet ikke inn i de andre kategoriene. Dette er de linjene som man ofrer minst ved å sløyfe.

A.3 Generelle regler og unntak til disse

Alle linjene har noen klare fellestrekk. De starter med et tegn som beskriver hendelsestypen, etterfulgt av ett eller flere mellomrom; så kommer tidspunktet hendelsen skjedde på, der det siste sifferet i tidspunktet alltid står 5 plasser etter det første tegnet. Så kommer det et mellomrom, et kolon (:), og et nytt mellomrom. Så står det hvilken plattform hendelsen gjelder. Starten på en logglinje vil dermed typisk se slik ut:

B 305 : Platform 213 .....

Det er noen unntak til disse generelle reglene. For det første er det noen få linjer som ikke kommer inn under noen av hendelsestypene over. Det er den første linjen, de to siste linjene, og eventuelt linjen der det fortelles at alle plattformene på en side er slått ut (eller har utført alle ordrene i dreieboka). Disse har et mellomrom i stedet for det første tegnet, og linjen gjelder ikke noen spesiell plattform. For eksempel ser den siste linjen i loggfilen alltid slik ut:

412 : Stoping simulation no 1

Det andre unntaket gjelder en bestemt linje som forteller at en plattform er truffet av et eller annet, og ser slik ut:

135 : Weapon : Calculating effect on target 102

Bortsett fra disse unntakene, følger de aller fleste logglinjene mønsteret som er beskrevet over. De aller fleste har også den egenskapen at etter plattformnummeret kommer et mellomrom, fulgt av posisjonen til plattformen på det aktuelle tidspunktet. Posisjonen består av 6 sifre, et kolon, og 7 sifre. De første 6 sifrene er UTM-koordinatene i x-retning, de 7 siste i y-retning. Etter dette følger et kolon, og så selve ordren som utføres. Dersom en logglinje **IKKE** følger denne regelen, vil det bli spesifisert; alle andre linjer følger dette mønsteret.

A.4 Faktisk utseende til forskjellige logglinjer

A.4.1 Basistypen:

Tegnet for basis-typer er B. Basistyper er logg-linjer som må være med for at SP og KL skal kunne fungere, og representerer dermed et minimum av informasjon. Denne typen er den eneste som er absolutt nødvendig, og dersom det er behov for rask postprosessering, kan man klare seg med å bare velge disse linjene (simuleringene i SIMBA tar fortsatt like lang tid). Et eksempel på en basis-linje:

B 0 : Platform 214 423219:7669586 :EXECUTING WAIT until 140

De forskjellige mulige logg-linjene av basistypen er:

- START. Dette er den første ordren til alle direkteskytende plattformer.
- STOP. Plattformen er ferdige med alle sine ordre, og fjernes fra simuleringen.
- Weapon x ADD_AMMO y. Plattformens våpen nr. x får y ammunisjonsenheter.
- WAIT until x. Betyr at plattformen skal vente til tidspunktet x før den utfører neste ordre. Den får beskyte eventuelle mål som detekteres.
- END WAIT. Plattformen er nå ferdig med å vente, og klar til å utføre neste ordre.
- WAIT_FOR_SHOTS x. Plattformen skal vente til den har avfyrt x skudd før den utfører neste ordre.
- HIDE until x. Betyr at plattformen skal gå i dekning, og vente til tidspunktet x med å utføre neste ordre.
- END HIDE. Plattformen er ferdig med å være i skjul, og er klar til å utføre neste ordre.
- MOVETO x:y: z. Plattformen starter å bevege seg mot posisjonen med UTM-koordinater x og y. Dette tilsvarer en bevegelse i en vinkel z i forhold til øst.
- SET_SPEED x. Hastigheten som plattformen beveger seg i. Denne linja kommer alltid etter ”MOVETO”-linja.
- GOTO x:y. Plattformen flyttes instantant til det forhåndsbestemte punktet, med UTM-koordinater x og y.
- SET_FLAG x. Setter flagget x til TRUE. Dette flagget kan brukes til å trigge hendelser hos andre plattformer.
- RESET_FLAG x. Setter flagget x til FALSE.
- WAIT_FOR_FLAG x. Plattformen venter til flagget x er TRUE før den utfører neste ordre.

- EXECUTING ART x. Artilleriild lander i denne posisjonen. Posisjonen er **ikke** artilleriets posisjon, men senteret i artilleriets nedslagsfelt. x er vinkelen som skytsene skyter i (positiv omløpsretning, 0^o=øst). Dette har betydning for bomavstand og spredning.
- END EXECUTE. Artilleri er ferdig med å skyte. Posisjonen foran har samme betydning som over.
- DAMAGE RADIUS r. Dette betyr at den aktuelle granaten kan slå ut mål innenfor en radius r. For smarte granater tilsvarer dette søkeområdet.
- Weapon x SHOOTING mi y. Plattform A, som er en direkteskytende plattform, skyter mot plattform B med våpen nr. x. Dette er våpen x sitt y'te skudd.
- SHOOTING tube # x. Plattform A, som er artilleri, skyter med rør (skyts) nr x.
- A MISSED B. Skuddet fra plattform A bommet på plattform B. For smart artilleri betyr det enten at det ikke var mål innenfor søkeområdet, eller at den ikke oppdaget noen av målene.
- A HIT B. Skuddet fra plattform A traff plattform B, men gjorde ingen ekstra skade.
- A M-KILLED B. Skuddet fra plattform A førte til en "mobility-kill" på plattform B.
- A F-KILLED B. Skuddet fra plattform A førte til en "firepower-kill" på plattform B.
- B TOTALKILLED: Plattform B ble slått ut fullstendig.
- B ALREADY TKILLED: Kommer eventuelt etter effekten av et skudd, og forteller at plattform B allerede var slått ut før skuddet traff.

A.4.2 Deteksjonstypen:

Tegnet for deteksjonstypen er D. Denne typen inneholder informasjon om scanning og deteksjon. Denne informasjonen etterfølges alltid av informasjon om plattformen som "deteksjonstypen" utføres på. Et eksempel:

D 750 : Platform 101 461900:7671272 :Obsdev 1 (EYE) DETECTED platform 202 462019:7673630, dist2tg 2361

Det er ikke linjeskift i logg-filen selv om linja er lang. Ordet "dist2tg" betyr "distance to target", og tallet etter dette ordet er avstanden mellom plattformene. La oss kalle plattformen som utfører hendelsen for A, og den andre for B. De forskjellige mulighetene er:

- Obsdev x SCANNING FOR. Det er blitt mulig for plattform A sitt observasjonsmiddel nr. x å detektere plattform B.
- Obsdev x STOP SCAN of. Det er blitt umulig for plattform A sitt observasjonsmiddel nr. x å detektere plattform B.
- Obsdev x DETECTED B. Plattform A sitt observasjonsmiddel nr. x har oppdaget plattform B.
- Obsdev x IDENTIFIED. Plattform A sitt observasjonsmiddel nr. x har identifisert plattform B som et mål.
- SELECTED platform B. Plattform A har valgt plattform B som sitt mål.
- SELECTED weapon x. Plattform A har valgt våpen nr. x. Denne linja inneholder **ikke** informasjon om plattform B.
- Obsdev x DETECTED SMOKE B. Plattform A sitt observasjonsmiddel nr. x har detektert skuddsignaturen til plattform B.

- Obsdev x EXIT B. Plattform A kan ikke lenger skyte mot plattform B på grunnlag av informasjon fra observasjonsmiddel nr. x. Dette skjer et tilfeldig antall sekunder etter at observasjonsmiddelet ikke lenger kan se plattform B.

Et spesialtilfelle er når et mål som kun er detektert gjennom røyk blir valgt som mål. I stedet for plattformnummer og posisjon, kommer da meldingen ”SELECTED SMOKETARGET”. Avstanden til målet blir fortsatt oppgitt.

A.4.3 Ildprosedyre-typen:

Tegnet for ildprosedyrer er F. Eksempel:

F 790 : Platform 201 461976:7673551 :Weapon 1 GUN LOADING

De forskjellige ildprosedyrene er:

- Weapon x Type LOADING. Plattformens våpen nr. x (av type Type) begynner å lade.
- Weapon x Type LOADED. Plattformens våpen nr. x er ferdig med å lade.
- Weapon x Type AIMING. Plattformens våpen nr. x begynner å sikte/klargjøre skudd.
- Weapon x Type AIMED. Plattformens våpen nr. x er ferdig med å sikte, og klar til å skyte.

A.4.4 Missilbevegelser:

Tegnet for missilbevegelser er M, og de forteller om hvor og hvor langt fra målet et bestemt missil er. Disse ser alltid ut på samme måte, det er bare missilnummeret og avstanden til målet som endres. Linjene ser slik ut:

M 773 : Platform 104 : FF : Missile 1 at 461936:7671585 dist2tg : 2059m

Det er altså ingen informasjon om hvilken plattform missilet stammer fra.

A.4.5 Siktelinje-typen:

Tegnet for Siktelinje-typer er V. Disse beskriver hvilket område en plattform leter etter fiender i. Etter plattformnummeret følger posisjonen, mellomrom, kolon og ”hendelsen”. Eksempel:

V 0 : Platform 232 461002:7674642 :Obsdev 1 SET_VIEW_SECTOR 20

De forskjellige ”siktelinjehendelsene” er:

- Obsdev x SET_VIEW_DIRECTION x. Plattformens observasjonsmiddel nr. x settes til å observere i en retning som er x grader i forhold til plattformens retning.
- Obsdev x SET_VIEW_SECTOR y. y er her den sektoren som plattformens observasjonsmiddel nr. x observerer i. Jo mindre denne er, dess lettere er det å detektere noe i sektoren.
- Obsdev x SET_ELEVATION y. Som over, men y er nå sektoren i planet normalt på bakken. Denne har kun betydning for deteksjonssannsynligheten; plattformens observasjonsmiddel nr. x kan detektere i alle høyder.
- Obsdev x SET_VIEW_RANGE y. y forteller her hvor langt plattformens observasjonsmiddel nr. x ”ser”. Ingenting som ligger utenfor denne rekkevidden kan detekteres av dette observasjonsmiddelet.
- Obsdev a SET_VIEW_POINT x y. Denne linja forteller at siktemiddelet a midlertidig ser mot posisjonen x, y. Den ser altså ikke lenger i en bestemt retning, men mot et bestemt punkt.

A.4.6 Resten:

Tegnet for de resterende logglinjene er #. Disse linjene er av begrenset interesse for de fleste brukere.

De siste linjene er:

- SET_FIRE_DENIED x. Forbyr plattformen å skyte til den får kontraordre. x er her navnet på flagget som settes. Forbudet gjelder plattformen, og ikke kun et enkelt våpen.
- SET_FIRE_ALLOWED x. Tillater plattformen å skyte.

I tillegg kommer det to linjer når en plattform blir truffet av et skudd / missil. De ser slik ut:

```
#  125 : Platform 202 : Searching for PKILL : Target = JAVELIN
#  125 : Weapon : Calculating effect on target 102
```

Dette betyr at et skudd avfyrt fra plattform 202 har truffet plattform 102, som er en JAVELIN type.

A.5 Spesielle bemerkninger til artilleri

Artilleri/BK kan ikke slås ut, og har ingen posisjon. Posisjonen som gis i forbindelse med ”EXECUTING ART” er sentrum i artilleriets nedslagsfelt. Her er bomavstanden tatt med. Posisjonen som oppgis i forbindelse med ”SHOOTING tube x”, er posisjonen der en granat faktisk lander. Denne er modifisert i henhold til spredningen og det aktuelle skytsets siktepunkt i forhold til nedslagsfeltets sentrum.

Litteratur

(1) HALSØR Marius, EIDE Morten (2003): SIMBA - Brukerveiledning, 01712, Ugradert