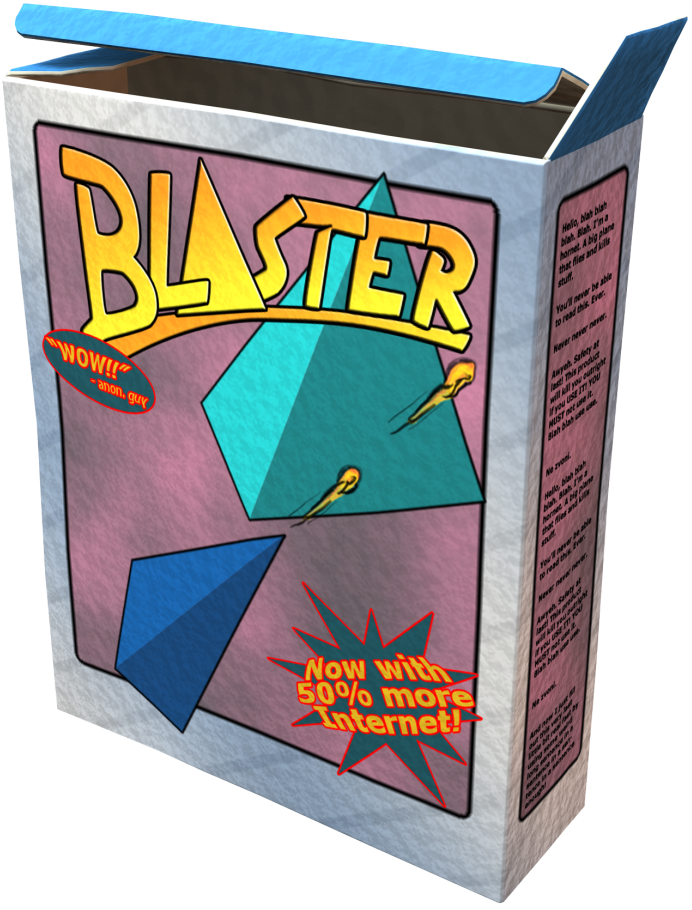
Multiplayer netværksspil i C++

doktor@dyregod.dk — **Ulf Holm Nielsen**
tnjr@ruc.dk — **Thomas Riisbjerg**
mads@danquah.dk — **Mads Danquah**
tkrogh@ruc.dk — **Troels Krogh**

Vejleder:

bjchr@ruc.dk — **Bjørn Christensen**

24. december 2002



Roskilde Universitetscenter
Datalogisk Institut, Datalogi OB
1. modul

Abstract

Projektets formål har været at designe og implementere et multiplayer netværks spil samt en bane editor til dette spil. Det var et krav at spillet og editoren skulle kunne køre på flere computerplatforme, nemlig MacOS X og Windows. For at kunne køre på flere platforme blev spillet programmeret C++, udelukkende ved brug af API der er tilgængelige på flere platforme. Til grafik er benyttet OpenGL og SDL, til netværk er benyttet SDL_net. Editoren blev implementeret i Java.

Spillet er designet godt men implementeringen virker ikke som planlagt. Netværks delen virker kun delvis, pga tidspres blev denne aldrig færdig. Selve grafik delen og resten af spillet fungerer som planlagt. Editoren er implementeret og designet på tilfredsstillende vis og fungerer præcis som ventet.

Den medfølgende CD indeholder al kildeteskt samt programmet i binær form.

The goal of this paper was to design and implement a cross-platform multiplayer game along with a level editor. The game and editor were required to run on both MacOS X and Windows.

In order to implement the game across several platforms the game was programmed in C++ using APIs available on both platforms. The graphics were drawn using OpenGL and SDL, and SDL_net was used to implement the networking code. In order to implement a cross-platform GUI, the editor was implemented in Java.

The game is well-designed, however the implementation does not work as expected. The networking code only partially works and was not finished due to time restraints. The rest of the program works as expected. The editor was implemented to a satisfactory degree and works as expected.

The accompanying CD contains the full source code as well as compiled versions of the programs.

Indhold

Figurer

Kode-lister

Forord

Denne rapport er udarbejdet som led i 1. moduls projekt på Datalogi OB ved Roskilde Universitetscenter. Rapporten er udarbejdet af Ulf Holm Nielsen, Thomas Riisbjerg, Mads Danquah og Troels Krogh.

Forsidebilledet forestiller en fiktiv æske til spillet Blaster og er lavet af Jon Lauridsen.

Programkoden er kommenteret på engelsk i koden, men beskrevet på dansk i rapporten.

Rapporten er sat i LaTeX, alle figurer er udarbejdet af projektgruppen.

Rapporten og programkoden er også tilgængelig i elektronisk form på den medfølgende CD samt på følgende adresse:

http://www.dyregod.dk/blaster/

Læsevejledning

Målgruppe

Denne rapport er henvendt til folk med interesse for implementering af computerspil.

Der forudsættes matematikkundskaber svarende til A niveau fra gymnasiet, og et kendskab til lineær algebra vil være en klar fordel. Der forudsættes datalogiske kundskaber svarende til videregående kurser i programmering og design på universitetsniveau. Desuden forudsættes kendskab til Java og C++ samt grundlæggende computergrafik.

Rapport opbygning

Indledning - handler om gruppens motivation for at vælge projektet sammen med en kort introduktion til selve projektet.

Problemanalyse - forsøger at anskueliggøre for læseren hvilke overvejelser og beslutninger der ligger bag konstruktionen af de to programmet.

Derefter splitter rapporten op i to. En del beskæftiger sig med klienten og serveren, den anden med editoren. Hver del indeholder følgende afsnit:

Kravspecifikation - opstiller krav for hvad programmet skal kunne

Design - gennemgår det overordnede design af programmet og overvejelser der har ledt til det

Implementation - omhandler hvordan designet er implementeret

Afprøvning - undersøger om programmet virker i henhold til kravspecifikationen

Herefter samler rapporten sig igen.

Diskussion - vurderer og diskuterer programmerne

Konklusion

Perspektivering

Afsnit 1

Indledning

Vi har i denne rapport valgt at udvikle et netværksbaseret computerspil. Selve idéen til at lave et spil kommer fra en fælles interesse for computerspil som igennem mange år har beslaglagt mange timer af vores liv. Det kunne derfor være interessant at opbygge et spil fra bunden og lære hvorledes spil egentlig fungerer. Vi har valgt at beskæftige os med et netværksbaseret "alle mod alle" skydespil. Det er en af de mest populære genrer indenfor spil i dag.

Overordnet går denne slags spil ud på at hver person styrer en spiller rundt på skærmen. Når 2 eller flere spillere møder hinanden, gælder det om at tilintetgøre hinanden. Fordelen ved denne slags spil er, at det er overkommeligt at nå på den givne tid da, der i modsætning til et singleplayer spil ikke skal udarbejdes et langt historieforløb som spilleren skal igennem for gennemføre spillet. I stedet kan spillere komme og gå som de har lyst. I denne slags spil er det konkurrencen om at score flest point, der er lagt vægt på, fremfo historien. Mere specifikt skal spillet tage udgangspunkt i de gamle Amiga spil Turbo Raketti[Aho01] og Gravity Force, hvor det gælder om at styre et rumskib rundt på skærmen og skyde modspillere.

Selve spillet bliver kodet i C++, da det har været en af ambitionerne i projektet at opnå større kendskab til C++. Til spillet hører desuden en bane-editor kodet i Java. Der var for så vidt ikke noget i vejen med at kode både spil og editor i Java. Men C++ delen af projektet har været en vigtig del af motivationen for at skrive projektet. En af ulemperne ved at bruge C++ er at man mangler den platformuafhænighed som Java tilbyder. For at komme uden om dette problem, var det nødvendigt at finde nogle biblioteker, der kunne håndtere platformuafhængigheden for C++.

Udviklingen af programmet er sket iterativt. Dette sikrede, at der altid var en fungerende version af programmet at falde tilbage på. Således blev der først udviklet en version, hvor spilleren kunne flyve rundt alene. På dette grundlag var det muligt at gå i gang med en version, hvor flere spillere kunne deltage.

Da spillet skulle udvikles af 4 personer var nødvendigt at udviklingen kunne ske uden at samtlige personer skulle sidde i samme lokale når der skulle kodes. For at opnå denne frihed er versionerings systemet Concurrent Versions System (herefter CVS) blevet benyttet. Med dette værktøj i hånden har det været muligt at styre projektets filer fra et centralt sted. Det har været muligt at se hvilke brugere, der har lavet opdateringen og hvad de har opdateret. Det har været med at at sikre at fungerende kildekode ikke blev overskrevet eller slettet. Des-

uden er CVS opbygget så det ikke er muligt at slette filer, men i stedet bliver filerne versioneret. Det vil sige at det altid er muligt at finde en ældre version af filen frem, selvom den ved et uheld er blevet overskrevet med uhensigtsmæssige rettelser.

CVS er også blevet benyttet under selve rapportskrivningen til at versionere de forskellige tesktdokumenter. Brug af CVS til rapportskrivningen giver præcis de samme fordele som under kode delen.

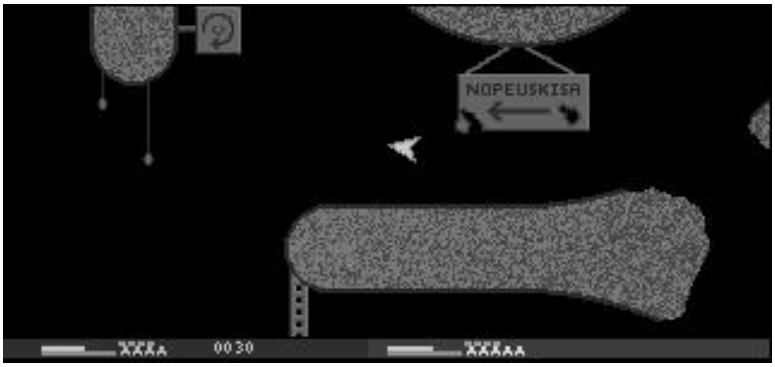
Del I

Problemanalyse

Afsnit 2

Inspirationskilder

Blaster baserer sig som nævnt hovedsagelig på minderne fra to gamle Amiga spil: *TurboRaketti 2* og *Gravity Force*. Begge går ud på at spilleren flyver rundt i et rumskib og skal skyde en anden spiller. Skærmen er delt i to, hvor de to spillere vises i hver sin del af skærmen.



Figur 2.1: Turbo Raketti 2

Begge spil bygger på det koncept, at man starter fra en base og skal skyde det andet fly med en begrænset mængde brændstof og ammunition, som kan genopfyldes ved at lande på en vilkårlig base.

Netværksfunktionaliteten er inspireret af nyere netværks-baserede spil, hvor en spiller forbinder til en central server og kommer ind i et igangværende spil med de andre spillere, der eventuelt måtte være forbundet til serveren.

Afsnit 3

Gameplay

3.1 Gameplay

Spillet kommer til at udspille sig i et 2d miljø, hvor små rumskibe skal udkæmpe en "alle mod alle" kamp. Banerne, hvor slagene bliver udkæmpet, er udformet som et sammenhængende netværk af gange, hvor der rundt omkring vil være mulighed for at rumskibene kan lande og tanke op.

Et rumskib består af en motor, et skjold og en kanon. Motoren driver flyvet frem når brugeren beder om det og forbruger hverved brændstof. Hvis skibet skulle løbe tør for brændstof, vil det bliver trukket i den retning som den gravitationelle kraft peger.

Skjoldet, som skibet er udstyret med, skal beskytte skibet mod skud fra andre spillere og desuden vil skjoldet forhindre, at skibet bliver ødelagt ved sammenstød med banens kanter og andre skibe. Skjoldet opretholdes af en energikilde i skibet, som langsomt drænes efterhånden som skibet udsættes for sammenstød.

Skibets kanon kan affyre skud med et kort interval, indtil kanonen løber tør for ammunition.

For at være fuldt kampdygtigt skal skibet bruge energi til sit skjold, brændstof til motoren og ammunition til sin kanon. Når et nyt spil startes, vil skibet være tanket fuldt op med disse 3 ting. Efterhånden som spillet skrider frem, vil skibets ressourcer blive brugt og det vil være nødvendigt for skibet at tanke op. Dette foregår på platforme, der er placeret rundt omkring på banerne. Det er de samme platforme, som skibene starter fra, når spillet begynder. Det er også disse platforme skibene starter fra, når en spiller dør.

Der er 3 måder hvorpå en spiller kan dø:

- Skibet mister skjoldet ved enten kollisioner med væg, skib eller skud.
- Skibet kan blive suget ind i et sort hul.
- Skibet løber tør for brændstof og falder ned.

De første 5 sekunder efter at en spiller er blevet gendannet er spilleren usårlig. Dette indikeres på skærmen.

Væggene i gangene vil være elastiske, hvilket vil sige, at rumskibe, der flyver ind i en væg, vil blive frastødt igen under tab af energi afhængigt af pågældende

overflade.

Når spilleren accelererer rumskibet vil hastigheden af skibet øges indtil den når et maksimum.

3.2 Netværk

Som nævnt i det foregående afsnit, skal det være muligt for flere spillere at spille spillet samtidigt. Før dette kan lade sig gøre skal det være muligt for hver spiller at holde sig opdateret om, hvad de andre spillere foretager sig. Spillet skal holde sig opdateret via en række beskeder, der bliver sendt via et netværk, som spillet kobler op til ved spillets start. Disse beskeder indeholder information så som:

- En spillers position, orientering og hastighed.
- En spiller er død.
- En spiller er genopstået.
- En spiller har affyret et skud.
- En spiller har forladt spillet.
- En ny spiller er på vej ind i spillet.
- En spillers status (point, energi, brændstof...).

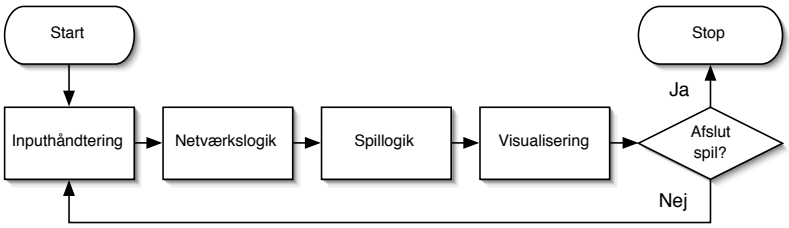
Der vil være en konstant strøm af beskeder i netværket da spilleren hele tiden skal opdateres med de andre spilleres gøre og laden. Derudover kommer en spiller til selv at kunne sende beskeder med sin egene data.

Afsnit 4

Komponenter i et spil

I dette afsnit identificeres de komponenter der indgår i et spil fra et programmeringssynspunkt.

Målet med projektet er at lave et actionspil. Spillet indeholder en hovedløkke der sørger for at opdatere spillet for derefter at tegne det på skærmen. Et enkelt gennemløb af hovedløkken kan deles ind i et par komponenter der er karakteristiske for computerspil: *inputhåndtering* fra brugeren, *netværkslogik* fra andre brugere eller en server, *intern spil logik*, og til sidst *visualisering* af spillet.



Figur 4.1: Flow chart for et eksemplarisk action spil

Ved inputhåndtering forstås, at spillet påvirkes af brugerens interaktion. Det kan være spilspecifikke ting som at påvirke spillerens bevægelse, eller mere programorienterede som at afslutte spillet.

Netværkslogikken håndterer information fra andre spillere. Her modtager spillet information om hvilke spillere der tilslutter og forlader spillet samt andre spilleres placering, antal point og lignende.

Spillogikken består af at opdatere spillets tilstand. Det kan bestå af at opdatere spillerens placering i banen (ud fra forrigt input), opdatere eventuelle fjenders placering, tildele spilleren point eller fratrække spilleren noget energi, undersøge om spilleren er stødt ind i noget i banen etc.

Visualiseringen af spillet består i at tegne spillet i dets vindue. Spillerens objekt skal tegnes såvel som eventuelle fjender. Banen skal tegnes. Hvis spillet har en baggrund foruden banen, skal denne også tegnes.

Afsnit 5

Centrale valg

I dette afsnit bliver der redegjort for hvilke overordnede valg der har formet udviklingen af Blaster.

5.1 Visualisering

Der er mange fremgangsmåder til at visualisere et spil på. Først er det vigtigt at identificere de forskellige typer af elementer der skal tegnes i Blaster. Elementer som rumskibe og skud er ofte ret små i forhold til størrelsen på skærmen eller vinduet som spillet tegnes i. Omvendt er den verden som spilleren bevæger sig i, ofte større end skærmen, således at spilleren kun ser et lille udsnit af banen på et givet tidspunkt. Yderligere kan der være visse oplysninger som spilleren er interesseret i at se på skærmen: brændstof, antallet af points og så videre.

De to mest relevante visualiseringsmetoder i dette tilfælde er bitmaps og sprites, samt polygonbaseret grafik. Tidlige computerspil brugte hovedsageligt bitmaps, mens nye computerspil overvejende benytter sig af polygoner til at visualisere sig selv.

5.1.1 Bitmaps

Vælges den bitmaporienterede fremgangsmåde betyder det, at små elementer på skærmen tegnes som *sprites*, og større elementer på skærmen (verden) tegnes enten som et stort bitmap, eller som en række *tiles*.

En *sprite* er en lille rektangulær bitmap som har en størrelse, hvor hvert led er en andenpotens. Dette gør dem særligt hurtige at tegne for grafiksystemet. Selv om en sprite i sig selv er rektangulær, kan dele af den være defineret som gennemsigtige, således at en sprite kan fremstå runde eller hule. Dette vil ikke blive gennemgået nærmere.

En *tile* er en sprite der er tegnet således at den kan placeres ved siden af sig selv eller en anden sprite, således at de to sprites har en naturlig overgang og fremstår som én. En skov kan f.eks. repræsenteres ved at have en lille sprite der viser nogle træer, og så tegne flere af disse sprites ved siden af hinanden. Fordelen ved tiles er, at det er muligt at opbygge et større billede ud af små byggeklodser. Dette bl.a. er en fordel både hvad angår hukommelsesforbrug.

En stor ulempe ved sprites er, at der kræves en ny sprite for hver retning som rumskibet kan bevæge sig, da man normalt ikke rotere sprites. Det betyder man enten skal begrænse de retninger, som rumskibet har mulighed for at bevæge, eller også må man vælge den sprite er tættest på at repræsenterer den orientering som rumskibet har på det givne tidspunkt. Enten begrænses spillerens bevægelsesmuligheder, eller også accepteres det, at der godt kan være en uoverensstemmelse med den orientering, som skibets sprite repræsenterer i forhold til skibets egentlige orientering.

Grunden til at man undgår at rotere, eller på anden måde manipulere, sprites, er hastighed. Hvis man manipulere med en sprite før man tegner den, mister man en del af den hastighed man prøvede at vinde ved at bruge en sprite.

Uanset hvor meget det er muligt at reducere kravet til mængden af sprites og tiles vil der altid være et minumumsantal, der skal tegnes. Eksempelvis er det næppe acceptabelt med mindre end 8 mulige orienteringer for rumskibet, selv om antallet af unikke sprites kan reduceres ved brug af spejling, og af hensyn til variation i banen skal der også være et par tiles at vælge imellem. Dette stiller krav til at disse rent faktisk skal tegnes.

Et andet potentielt problem ved bitmaps er, at det ikke er ligetil at bestemme, hvad der sker, hvis et rumskib kolliderer med banen. Grunden til dette er at man normalt ikke ønsker at kollidere med selve spriten, men dens grafiske repræsentation. Normalt vil man f.kes. ikke kollidere med gennemsigtige dele af spriten. Men hvordan finder man nemt hældningen på en del af en bitmap? Hvad hvis der kun er en enkelt løsstående pixel som kolliderer - hvilken retning skal skibet da afbøjes i? Derfor er sprites ikke gode kandidater til detaljeret kollision.

5.1.2 Polygoner

Den anden metode er som nævnt at bruge polygoner til at repræsentere de forskellige elementer i banen. Når der arbejdes med polygoner er det forholdsvis nemt at beregne rumskibets nye bevægelsesretning, da siderne af en polygon består af veldefinerede linier. Ud fra disse er det let at beregne reflektionsvektoren ud fra det indkommende rumskibs hastighedsvektor.

Veldefinerede kollisioner er ikke den eneste fordel som polygoner har i forhold til bitmaps. Det er simpelt at rotere en polygon i en vilkårelig vinkel, da polygoner er defineret ud fra et antal punkter i et koordinatsystem, og disse kan transformeres.

Det er dog vigtigt at pointere, at det er lidt langsommere at rotere en polygons punkter og derefter tegne den, end at tegne en simpel sprite. Uden at gå for meget i detaljer, så tegnes en sprite ved at kopiere dens bitmap over i skærmens bitmap. For at tegne en polygon skal den først rasteriseres i et bitmap, og dette skal da kopieres ind i skærmens bitmap [vDSFJH].

De væsentligeste grunde til at polygonbaseret grafik er at foretrække frem for bitmaps er altså, at brugen af polygoner garanterer at elementerne i spillet kan tegnes med samme orientering som i modellen, samt at kollisioner mellem to polygoner i bevægelse umiddelbart er mere veldefineret end mellem to sprites.

5.2 Banen

Når det er vedtaget at bruge polygoner til at repræsentere grafikken i banen, skal der findes en metode til at definere en bane i spillet.

En mulighed er at indlæse filer fra et eksisterende modelleringsværktøj, som Maya eller 3D StudioMax. Disse programmer er designet til at lave avancerede 3D modeller, ofte opbyggede af polygoner. Dog indeholder sådanne programmers filformater overflødig data i forhold til de krav som Blaster stiller, og understøtter ikke nødvendigvis de egenskaber for polygoner som Blaster kræver. Desuden kan sådanne filformater være komplekse og udokumenterede. Derfor vurderes det at Blaster må have sit eget filformat til banerne.

Filformatet skal beskrive de polygoner der ligger i banen, såvel som de egenskaber polygonerne og banen måtte have. Dog bliver det nemt uoverskueligt at redigere en fil med mere end et par polygoner, hvorfor det vurderes at være nødvendigt at lave en lille editor der er i stand til at redigere baner med et grafisk interface.

5.3 Netværket

Et af målene for Blaster er som nævnt, at flere spillere skal kunne spille mod hinanden. Det betyder, at data skal sendes mellem spillere. Det præsenterer to problemer: Hvilken netværksmodel skal bruges til at sende data mellem spillerne, og hvilket indhold skal der være i de data der transmitteres mellem spillerne.

5.3.1 Netværksmodeller

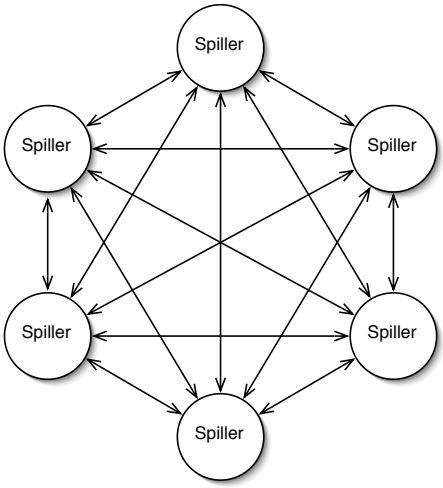
Netværksmodellen bestemmer hvordan information om spillets tilstand sendes rundt på netværket, samt hvem der er ansvarlig for at opdatere spillets tilstand.

Peer to peer

Den simpleste netværksmodel er *peer to peer* modellen. I denne model har alle spillere forbindelse til samtlige andre spillere. Når en spiller vil opdatere spillets tilstand med dens information, skal den sende information om sig selv til samtlige andre spillere på netværket. En spiller er altså ansvarlig for at fortælle alle de andre spillere om sine handlinger.

Der er en enkelt fordel, men mange ulemper ved peer to peer netværksmodellen. Fordi hver spiller har direkte forbindelse til hinanden, skal netværksdataen kun igennem ét led, hvilket giver en lav forsinkelse fra dataen bliver sendt til den bliver modtaget. Desværre bruger denne model meget båndbredde, og forbruget stiger eksponentielt for hver ny spiller. Når hver spiller sender og modtager data fra samtlige andre spillere, kræver n spillere $(n-1)!$ netværksforbindelser. Hvis en spiller antages at sende 100 bytes ti gange hvert sekund til hver spiller i et spil med 20 spillere, sendes altså $1000\frac{bytes}{s} * (20-1)! = \ 1.22 * 10^{17}\frac{bytes}{s}$. Der går nok mange år inden der findes netværk der kan håndtere en sådan belastning.

Fordi hver spiller fortæller de andre spillere om dens tilstand, er der potentiale for at snyde ved at sende falsk information omkring spilleren til de andre spillere.

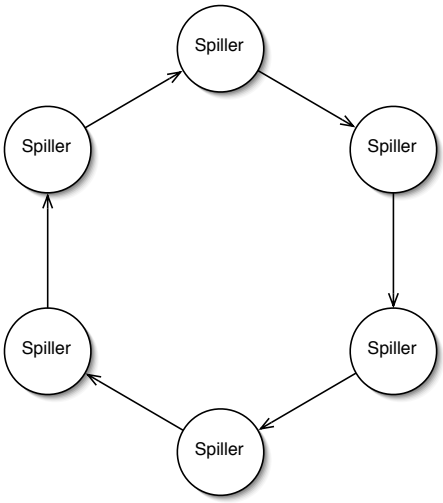


Figur 5.1: Peer to peer modellen

Der rejser sig også spørgsmål om hvad der sker hvis én spiller rammer en anden med skud: hvilken spiller er ansvarelig for at sørge for, at den ramte spiller bliver tildelt den skade den skal have? Hvordan forhindrer man at det bliver misbrugt? Hvis en spiller selv er ansvarlig for at håndtere skade, kan spilleren jo bare lade være med det. Omvendt, hvis andre spillere har ansvaret, hvad forhindrer så en spiller i at påstå at den pludselig har dræbt samtlige spillere i banen?

Ringmodellen

Ringmodellen er en forbedring af peer to peer modellen, som er opbygget efter samme princip som TokenRing netværk. Spillere organiseres i en ring, og hver spiller skiftes til at have lov til at ændre spillets tilstand. Mens en spiller venter på sin tur, gemmes input fra brugeren. Når det bliver spillerens tur, modtager den spillets tilstand fra den forrige spiller i ringen. Spilleren opdaterer spillets tilstand ud fra det input den har fået fra brugen, og giver turen videre til den næste spiller i ringen.



Figur 5.2: Ringmodellen

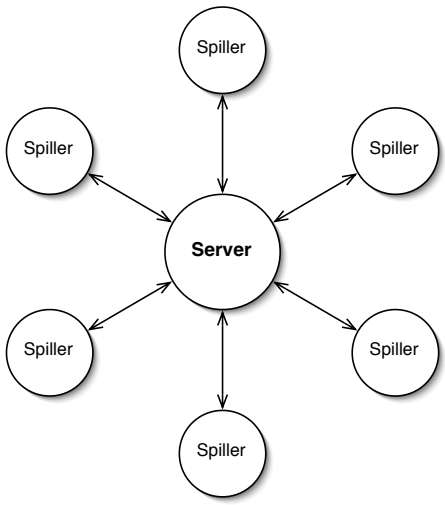
I forhold til peer to peer netværk er båndbreddeforbruget drastisk beskåret: på et givet tidspunkt er der kun to spillere der kommunikerer med hinanden. Desværre medfører ringmodellen en voldsom forsinkelse mellem hver gang en spiller har mulighed for at opdatere sin tilstand af spillet. Turen skal hele vejen rundt i ringen inden spilleren kan få sin tur igen. Forsinkelsen afhænger af antallet af spillere: hvis der er 19 andre spillere med i spillet, er spilleren nødt til at vente på at de 19 andre spillere opdaterer spillets tilstand. Hvis det antages, at spillet opdaterer lokalt 30 gange i sekundet hos hver spiller, tager det $0,033$ sekunder for en spiller at beregne en opdatering, hvilket inkluderer at læse fra og sende til netværket. En spiller skal altså vente $0.033s * 19 = \ 0.63s$, eller lidt over et halvt sekund mellem hver opdatering af spillet. Spillerens oplysninger om de andre spillere vil altså kun blive opdateret hvert halve sekund. Ringmodellen egner sig altså ikke til spil med ret mange spillere.

Der er stadig mulighed for snyd i ringbaserede netværk. Der er ikke noget der forhindrer en spiller i at opdatere spillets tilstand som det nu engang passer spilleren.

Klient-server modellen

Opbygningen af klient-servermodel har form som en stjerne med serveren i midten, omgivet af klienter (spillere). En givet spiller har kun forbindelse til serveren, og serveren har forbindelse til alle spillere i spillet.

Klient-servermodellen har mange fordele i forhold til de to andre modeller, og få ulemper. Båndbreddekravene til serveren er større end for en klient: det kræves at serveren skal håndtere forbindelser til samtlige forbundne spillere. Båndbreddekravene for en klient er blot forbindelsen til serveren.



Figur 5.3: Klient-server modellen

I en ideel klient-server situation behandler klienten ikke input fra brugeren, men sender det til serveren hvor det bliver behandlet. Serveren opdaterer spillets tilstand ud fra alle klienternes input, og sender visuel information om spillets nye tilstand til alle klienterne. Klienten fungerer altså stort set som en dum terminal hvor alle beregninger og opdateringer af spillets tilstand foregår på serveren, og klienten kun samler input og tegner spillet. Dette sætter automatisk

en begrænsning på muligheden for at sende falsk data, da en klient i så fald kun vil kunne snyde sig til hændelser som spilleren alligevel havde mulighed for at foretage sig. En spiller kan således ikke længere påstå at hoppe til den anden ende af banen på et øjeblik, eller dræbe spillere som den ellers ikke havde mulighed for at ramme.

Forsinkelsen mellem input fra brugeren og visuel reaktion er større end ved peer to peer, men mindre end ved brug af ringmodellen: inputtet fra brugren skal først sendes til serveren hvor den behandles. Først når serveren har opdateret spiltilstanden sendes der data til klienten så den kan opdatere skærmen.

Antallet af spillere påvirker ikke forsinkelsen mellem en klient og en server, og påvirker kun båndbreddekravene lineært, da en spiller så skal modtage flere beskeder fra serveren jo flere spillere der er med i spillet.

Af de nævnte netværksmodeller er klient-server modellen den der har de bedste egenskaber til et multiplayer spil: modellen er skalerbar og begrænser mulighederne for at snyde.

5.3.2 Protokollen

Klient-server netværksmodellen lægger som nævnt op til, at klienten sender input til serveren og får en form for grafikbeskrivelse tilbage fra serveren, om det er i bitmap form eller mere abstrakte tegneinstruktioner. I praksis giver dette en mærkbar forsinkelse mellem input og reaktion hos brugeren hvis der blot er en lille forsinkelse mellem klienten og serveren. Problemet er at klientens input skal igennem serveren før klienten kan tegne en opdateret repræsentation af spillet hvor brugerens input har haft konsekvenser.

Forsinkelsen kan ikke fjernes, men den kan til en vis grad skjules hvis klienten sender og modtager information om rumskibenes position, oritentering og hastighed i stedet for simpel input. Således er klienten i stand til at gætte hvor de andre spillere er henne mellem opdateringer fra serveren, og serveren er i stand til at gætte klientens placering og orientering imellem opdateringer fra klienten. Når en ny opdatering kommer fra serveren, overskriver klienten blot de andre skibes positioner m.m, ligeledes når serveren modtager en klients opdaterede position.

Således har klienten en kopi af spillets tilstand liggende lokalt, som den bruger til at visualisere spillet. Når brugeren bevæger sit rumskib, sker det i den lokale kopi af spillets tilstand. Det lokale rumskibs placering m.m bliver sendt til serveren med korte mellemrum, præcis som serveren sender information om alle de andre spillere til klienten. Den lokale kopi af spillets tilstand vil næsten altid være i let uoverensstemmelse med serverens opfattelse af spillets tilstand, men hvis serveren får lov til at beholde autoriteten har det ikke nogen større betydning. En spiller vil aldrig se præcis samme tilstand som serveren ser, men en approksimeret tilstand som skulle afvige meget lidt. Det er ikke så kritisk hvis nogle af rumskibene hopper en lille smule af og til, hvis klientens eget fly bevæger sig i en glidende bevægelse og reagerer med det samme.

5.4 Teknologivalg

Dette afsnit undersøger hvilke teknologier der skal vælges i designet af Blaster.

En kort opsummering af de valg der er foretaget hidtil:

Blaster skal visualiseres med polygoner. Der skal bruges en editor til at fremstille baner til Blaster, da filformatet er komplekst. Blaster skal kunne indlæse en bane som editoren har lavet. Både Blaster og editoren skal kunne afvikles på MacOS X og Windows, og gerne Linux hvis muligt. Selve Blaster skal bestå af en klient og en server der kan kommunikere med hinanden.

Kravet om, at både spillet såvel som editoren skal kunne afvikles på både Windows og MacOS X har stor betydning for hvilke teknologier, der er egnede til at implementere spillet og editoren. Det har desuden været et ønske fra forfatterne at forsøge at bruge sproget C++ hvis muligt. Dette ønske kom først og fremmest af en interesse for at lære et nyt programmeringssprog.

5.4.1 Valg af grafik API til spillet

Spillet skal kunne visualiseres med brug af Java2D eller OpenGL. Hvis spillet kun skulle afvikles på styresystemet Windows havde Direct 3D også været en mulighed, men er irrelevant, da Direct X ikke findes til Mac OS X.

Java2D og OpenGL tilbyder mange af de samme faciliteter. Begge APIer er i stand til at tegne polygoner, samt at bruge diverse transformationsmatricer til at orientere og placere polygonerne på skærmen. OpenGL har den store fordel, at det kan accelleres på hardware, hvilket gør det markant hurtigere end Java2D.

OpenGL kan bruges med både C og Java via 3. parts pakken GL4Java, og er implemeteret både i MacOS X og Windows, hvilket gør det til et naturligt valg til at visualisere spillet, uanset hvilket sprog spillet implementeres i.

5.4.2 Valg af programmeringssprog til spillet

Spillet skal kunne tegnes i et vindue eller på hele skærmen med OpenGL. En mulighed havde været at implementere selve programmet særskilt på MacOS X og Windows med henholdsvis Apples Cocoa API og Objective C programmeringssprog til Mac OS X og Microsofts Direct X API og C++ til Windows. Selve spillet kunne så implementeres som et C++ modul som begge programmer brugte til det egentlige indhold. Således ville Cocoa og Direct X stå for at oprette et vindue og håndtere input fra brugeren, og C++ modulet for at tegne spillet med OpenGL samt at behandle det input, som det modtager.

En anden mulighed havde været at bruge Java. Java har som tidligere nævnt adgang til OpenGL. Dog havde forfatterne et ønske om at benytte C++ til selve spillet hvis det var muligt.

En tredie mulighed var at bruge et 3. parts API og bibliotek ved navn SDL , som binder sig til henholdsvis Cocoa og DirectX, og i øvrigt har lignende bindinger til Linux. Brugen af SDL sikrer, at den samme C++ kode kan compiles og afvikles på både MacOS X og Windows. SDL håndterer skærmen og input, således at man slipper for at bekymre sig om styresystemsspecifikke problemer og kan koncentrere sig om selve spillet.

Sideløbende til SDL findes SDL_net, der ligesom SDL abstraherer mellem Cocoa og Direct X, abstraherer mellem begge styresystemers netværksimplementation.

SDL_net sørger dermed for, at netværkskoden ikke skal implementeres forskellige på MacOS X og Windows.

5.4.3 Valg af programmeringssprog til editoren

Editoren har nogle helt andre krav end selve spillet. Foruden at skulle kunne afvikles både på MacOS X og Windows, skal editoren have et grafisk brugerinterface (GUI) med vinduer, menuer og hvad der ellers hører til. SDL egner sig til at åbne tomme vinduer hvori et spil kan tegnes, men kan ikke håndtere egentlige GUI elementer.

Der findes en række 3. parts C eller C++ biblioteker som kan håndtere vinduer m.m. på både MacOS X og Windows, som f.eks. mxWindows. Dog vurderes det at Swing er nemmere at bruge, og da forfatterne alligevel har erfaring med Swing er det et naturligt valg at implementere editoren i Java.

5.4.4 Valg af filformat til banen

En bane skal kunne skrives fra editoren, som er et Java program, samt indlæses i spillet, som er et C++ program. Dette udelukker at bruge Javas Serializable interface til at flytte banen fra editoren til spillet, da det ikke giver mening at forsøge at indlæse en stream af Java objekter i et C++ program. Derfor må et tekst-baseret format bruges. Projektgruppen har tidligere erfaring med brug af XML dokumenter i Java via 3. parts pakken Electric XML , og fandt hurtigt en ligende pakke til C++ ved navn TinyXML . Da banens format i forvejen har en klar struktur, valgtes XML som det format der bruges til at flytte baner fra editoren ind i spillet.

5.4.5 3. parts værktøjer

I forbindelsen med udvikling af spil og editor, er der taget en række 3. parts biblioteker og værktøjer i brug. Disse er beskrevet rundt omkring i rapporten, her følger dog en liste samt kort beskrivelse over dem alle.

TinyXML er en simpel C++ XML-parser. TinyXML er OpenSource og kan hentes på www.grinninglizard.com/tinyxml

ElectricXML er et Java-bibliotek til at parse og manipulere XML-data. ElectricXML er gratis og kan hentes på www.themindelectric.com/exml/

SDL er et platformsuafhængigt multimedie-bibliotek. SDL er licenseret under LGPL og kan hentes gratis på www.libsdl.org

SDL_net er et platformsuafhængigt netværks-bibliotek. SDL_net er licenseret under GPL og kan hentes på www.libsdl.org/projects/SDL_net

OpenGL OpenGL er et standardiseret grafik-API. Mere information kan findes på www.opengl.org

Del II

Klient og Server

Afsnit 6

Kravspecifikation

6.1 Kravspecifikation

I kravspecifikationen vil de krav, der stilles til henholdvis klient og server blive remsset op. De vil i et senere kapitel blive gennemgået i større detaljer.

6.1.1 Overordnede krav til programmet

Der er nogle få overordnede krav programmet skal opfylde. Disse krav skal være gældende for både serveren og klienten.

- Spillet skal kunne spilles på flere forskellige operativ systemer.
- Spillet skal være uafhængig af processer arkitektur.
- spillet skal kunne spille over netværk, hvor kommunikationen skal foregår via en klient-server model.

6.1.2 Krav til serveren

- Serveren skal sende data til og modtage data via UDP fra klienter.
- Serveren skal oprette TCP forbindelser mellem klient og server.
- Alt UDP kommunikation skal foregå på port 1337.
- Indkommende TCP trafik skal accepteres på port 1337.
- Når en spiller forlader spillet skal server lukke TCP forbindelsen, og fjerne spilleren fra spillet.
- Serveren skal indeholde information om alle spillere.
- Serveren skal overvåge og opdatere samtlige spilleres tilstande.
- Serveren skal fungere som den autoritære i klient-server forholdet, serveren træffer alle beslutninger vedrørende spillet.
- Serveren skal være i stand til at indlæse en brugerdefineret bane.
- Server skal står for udregning af kollision..

- Serveren skal afgøre hvornår en spiller er død og i så fald fjerne spillerens skib.
- Serveren skal være i stand til at genoplive en spiller.
- Serveren skal implementere de i gameplay afsnitte nævnte spilelementer.¡

6.1.3 Krav til klienten

- Klienten skal modtage data fra serveren og sende data til serveren via UDP på port 1337
- Klienten skal operette en tcp forbindelse på port 1337.
- Klienten skal have indeholde information om alle andre spillere.
- Klienten skal med jævne mellemrum opdatere serveren med information om spillerens position, orientering, hastighed og desuden mængden af brændstof og skud.
- Klienten skal visualisere alle dele af spillet, såsom skibe, skud, bane og status elementer
- Klienten skal være i stand til at indlæse en brugerdefineret bane.
- Klienten skal kunne tegne polygoner på skærmen.
- Klienten skal være i stand til at håndtere input fra brugeren.
- Klienten skal være i stand til at manipulere spillerens skib ud fra inputtet.
- Klienten skal kunne tegne de definerede views på skærmen.
- Klienten skal implementere de i gameplay afsnitte nævnte spilelementer.

Afsnit 7

Designovervejelser

I dette afsnit vil de overvejelser, der er gjort vedrørende designet af klient og server bliver gennemgået. Først vil de generelle designovervejelser blive gennemgået, hvorefter de markante forskelle på klient og server vil blive gennemgået hver for sig.

7.1 Generelle designovervejser

Overordnet indeholder klienten og serveren meget fælles funktionalitet, hvilket indbyder til at genbruge den samme kode i begge programmer. Dette stiller dog visse krav til designet, da det skal være udformet generelt nok til at kunne håndtere begge programmers krav.

De skal begge:

- Sende og modtage beskeder.
- Indeholde information om banen og samtlige spillere.
- Være i stand til at manipulere de aktive objekter i spillet

Programmerne er bygget op omkring Model View Controller (MVC) mønstret.

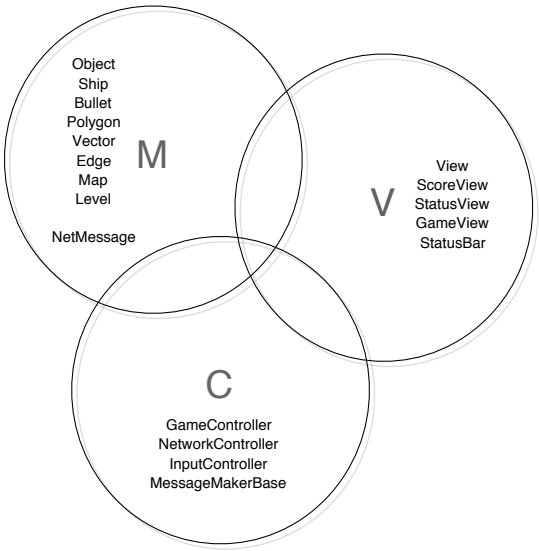
Modellen består af spillets data, dvs spillere, bane, skibe, polygoner, kanter og punkter. Samt information om spillets status. Disse komponenter er fælles for både klient og server.

View-delen består af klasser og metoder til at visualisere spillet. Samt klasser til at modtage input fra bruger og sende det videre til controller-delen. Da server hverken skal modtage input fra bruger eller vise noget på skærmen er denne del helt fraværende på serveren. View-delen vil derfor kun indgå i klient-programmet.

Controller-delen er så vidt muligt fælles for server og klient. Server står for for selve spillogikken dvs. den behandler spillerens handlinger. Klienten skal derimod håndtere input modtaget fra view delen. Desuden håndterer controller-delen hvert programs afvikling. For at klienten og server kan kommunikere over netværket på en hensigtsmæssig måde deler de et interface til netværkssystemet, som de begge implementerer og tilføjer deres egne specifikke metoder.

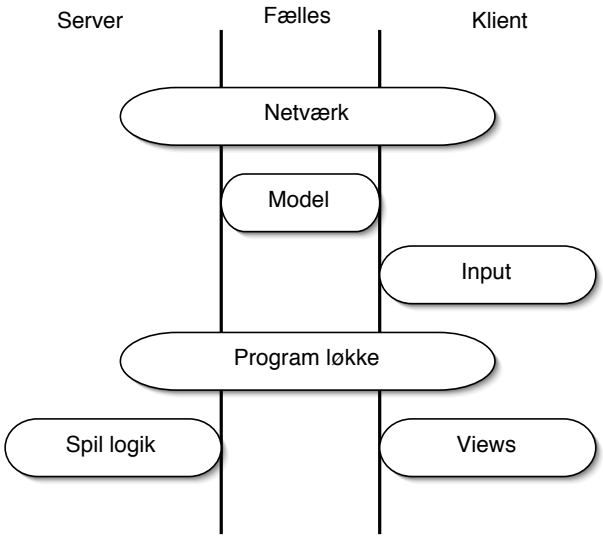
Figur 7.1 på side 31 viser en opdeling af de forskellige dele af klient og server i kasser: en for både model, view og controller. Som det fremgår af figuren er view

kun til stede i klienten, mens model og controller for største delens vedkommende deler funktionalitet.



Figur 7.1: Områder indelt efter MVC

De fælles områder er søgt illustreret på figur 7.2 på 31.



Figur 7.2: Oversigt over hvor delene hører til

7.2 Opbygning

7.2.1 Model

Modellen kan forholdsvis nemt danne basis for de klasser, der skal indgå i programmet. `Level` er den central klasse i modellen. Den indeholder en bane kaldet `Map` samt skibene (`Ship`) og deres skud (`Bullet`). Banen indeholder informationer om banens opbygning, størrelse etc.

Banens form repræsenteres af polygoner (`Polygon`). En polygon er opbygget af 3 kanter (`Edge`), der hver består af en vektor (`Vector`). Disse tre klasser bliver benyttet til at konstruere banen og skibet. Banen består af en lang række polygoner, der til sammen udgør den verden skibet kan bevæge sig rundt i. Skibet består også af polygoner. Helt simpelt behøver der kun være et enkelt, men der er intet i vejen for at sammensætte et skib af flere polygoner.

Da `Bullet` og `Ship` begge er bevægelige objekter skal de have mange af de samme egenskaber. Derfor nedarver de fra `Objekt` som indeholder data, der kan repræsentere hastighed, acceleration, position , orientering og ændringen i orientering.

For en oversigt over klasserne se figur **??** i bilag .

7.2.2 View

Funktionaliteten til at tegne enkelte objekter i modellen ligger i objekter selv. Således indeholder klasserne `Ship`, `Bullet` og `Map` funktionalitet til at tegne sig på skærmen. Derudover skal der på skærmen angives statusinformation om spillet. De infomationer, der er til til rådighed for spilleren er, resterende energi, skud, brændstof og antal point. Disse informationer er repræsenteret ved klassen `StatusView`. Endvidere er det muligt for spilleren at orientere sig om hvorledes stilling er i spillet. Dette foregår i `scoreview`, hvor man efter ønske kan blive præsenteret for en liste af samtlige spillere og deres point antal.

Da det aldrig er til at vide, hvilke andre views det kunne være hensigtsmæssigt at anvende i spillet, er view-delen opbygget efter designmønstret decorator som defineret i [Gam94]. Ved at anvende dette mønster er det nemt at opbygge et system til at håndtere udvidelse af informationerne på skærmen.

Inputhåndteringen skal med de valgte API'er foregår centralt. Derfor konstrueres en klasse, der for hver frame, der tegnes indeholder information om hvilke former for interaktion brugeren har udført. Denne klasse kaldes `InputController`

For en oversigt over klasserne se figur A.7 i bilag .

7.2.3 Controller

Controller-delen består af følgende dele: Netværk, spil logik og selve afviklingen af programløkken. De fælles dele af netværksfunktionaliteten er lagt i klassen `NetWorkcontroller`. Klienten og serveren kan nedarve fra denne klasse og tilføje den funtionalitet, der er specifik for dem.

Spillogikken, der ligeledes er forskellig for både server og klient, håndteres på samme måde som netværket, dvs der nedarves fra en fælles klasse. Denne klasse kalds `GameController`.

Programløkken findes for serveren i `ServerController` og for klienter i `ClientController`. Disse er ansvarlige for al opsætning, såsom at gøre skærmen klar til at tegne på i klientens tilfælde, starte netværk etc. For hver opdatering af spillet sørger controllerne for at håndtere input, opdatere spillets tilstand og i klientens tilfælde sørger `ClientController` for at view tegner spillet.

For en oversigt over klasserne se figur A.2 i bilag .

En samlet oversigt over delene i både klient og server kan ses på figur A.1 i bilag A.

Afsnit 8

Implementering af spillet

I det følgende vil implementering af Blaster blive præsenteret. Blaster er opdelt i to programmer: klienten og serveren. Disse deler umiddelbart meget kode, så hvis ikke andet er angivet bruges samme kode i serveren såvel som klienten.

Yderligere er afsnittet delt op i to dele: en del der beskriver de klasser, der bruges i selve spillet, samt en del, der beskriver de klasser der bruges til netværkskommunikation.

8.1 Spillet

8.1.1 ClientController

`ClientController` er indgangsklassen til klienten. `run` metoden indeholder selve programløkken som driver hele spillet indtil det afsluttes. Klassen sørger overordnet for, at beskeder bliver sendt og modtaget over netværket, at inputtet fra brugeren håndteres, at klientens spil-tilstand opdateres samt at klienten får tegnet spillet på skærmen. `run` metoden er skitseret i liste 8.1. Foruden `run` metoden indholder klassen en række af `View`s, der bruges til at tegne spillet. `ClientController` er implementeret som en singleton da der kun vil eksistere en enkelt instans af klassen på et tidspunkt.

Listing 8.1: Pseudo-kode for klientens programløkke

```
forbind til server
opret lokal spiller

indtil spillet afslutter:
   håndter input
   håndter beskeder fra serveren
   opdater spillets tilstand
   tegn spillet i skærmen
   send beskeder til serveren
loop
```

8.1.2 ServerController

`ServerController` er indgangsklassen til serveren. Den er baseret på `ClientController`, men tager sig udelukkende af at sende og modtage beskeder samt at opdatere spillets tilstand. Serveren skal altså ikke bekymre sig om at håndtere input fra en bruger eller at tegne spillet. `ServerController`s run metode er skitseret i liste 8.2.

Listing 8.2: Pseudo-kode for serverens programløkke

```
start server

indtil spillet afslutter:
   check for nye klienter
   håndter beskeder fra klienter
   opdater spillets tilstand
   send beskeder til klienter
loop
```

8.2 GameController

`GameController` klassen håndterer spillets tilstand. Det vil sige, at den sørger for at holde alle spillere opdaterer deres tilstand så vel som at sørge for at alle objekter opdaterer deres position og hastighed. Desuden undersøger `GameController` om forskellige objekter kolliderer med hinanden inde i spillet. `think` metoden er skitseret i liste 8.3.

Listing 8.3: Nuværende pseudo-kode for spillogikken

```
opdater alle spillere

opdater alle skibe, slet eventuelle døde

for alle skibe
   hvis skibet kolliderer med banen
   beskadig skibet og inverter
      hastighedsvektoren

opdater alle skud, slet eventuelle døde

undersøg om skud kolliderer med banen

undersøg om skud rammer skibe
```

Det er vitgigt at bemærke, at `GameController` ikke blev færdigimplementeret:

- `GameController` mangler at blive delt op i en `ClientGameController` og `ServerGameController` klasse på samme måde som `ClientNetworkController` og `ServerNetworkController`, og ved samme lejlighed skal think metoden refaktoreres til flere mindre metoder. Årsagen er, at klienten ikke

skal håndtere nogen form for kollision eller skade, den skal kun opdatere spillere, skibe og skud. Se liste 8.4.

- Kollisioner bliver ikke håndteret korrekt i den nuværende implementation af `GameController`. Da den nuværende kollisionsalgoritme kun er i stand til at opdage hvis to polygoner overlapper, er det kun muligt at invertere skibets hastighedsvektor ved kollision. Havde det været muligt at finde kant på det fremmede polygon som skibet kolliderer med, havde det været trivielt at finde reflektionsvektoren ud fra skibets hastighedsvektor og kantens normal. Se liste 8.5.
- Spillogikken er slet ikke koblet sammen med netværksbeskederne. Det var meningen, at `think` metoden skulle sende beskeder til henholdsvis alle klienterne eller serveren når visse begivenheder fandt sted. Dette kunne f.eks. være jævnligt at sende `ShipUpdate` beskeder til alle klienter, eller at sende en `PlayerKilled` besked til alle klienter når en spiller døde i `ServerGameController`.

Listing 8.4: Ønskede pseudo-kode for klient spillogikken

```
opdater alle spillere

opdater alle skibe, slet eventuelle døde

opdater alle skud, slet eventuelle døde
```

Listing 8.5: Ønskede pseudo-kode for server spillogikken

```
opdater alle spillere

opdater alle skibe, slet eventuelle døde
send ændrede skibspositioner til klienterne

for alle skibe
   hvis skibet kolliderer med banen
   find den kant som skibet kolliderer med
   sæt skibets hastighedsvektor lig med
      reflektionsvektoren

opdater alle skud, slet eventuelle døde
send nye skud til klienterne

undersøg om skud kolliderer med banen

undersøg om skud rammer skibe
```

8.2.1 View

View klassen og dens subklasser er ansvarelige for alt visualisering af spillet på klienten.

View er implementeret som en decorator[Gam94], så den har

GameView

`GameView` er ansvarelig for at tegne de objekter der eksisterer i `Level`. `drawContent` metoden i `GameView` er skitseret i liste 8.6

Listing 8.6: Pseudo-kode for tegning af GameView

```
centrer kameraet på klientens skib

tegn banen

tegn alle skud

tegn alle skibe
```

8.2.2 Level

Level indeholder de objekter der eksisterer inde i spillet: en liste af skibe, en liste af skud samt en bane.

8.2.3 Object

`Object` er en abstrakt baseklasse for alle objekter i banen. `Object` indeholder oplysninger om objektets placering, orientering, hastighed og acceleration. Et objekt indeholder yderligere `get` og `set` metoder til `owner` feltet, der henviser til hvilken spiller der ejer objektet i form af en spiller id, som refererer til de spillere der er gemt i `GameController`. `Object` indeholder også metoden `kill` der sætter et flag i objektet, således at det bliver slettet af `GameController`. `think` metoden opdaterer objektets position ud fra simpel mekanisk bevægelse med konstant acceleration. Klasserne `Ship` og `Bullet` nedarver fra `Object`. `Object` har også en abstrakt `collides` metode, som `Ship` og `Bullet` implementerer hver for sig.

Listing 8.7: Pseudo-kode for think i Object

```
hastighed = hastighed + acceleration + Δt

begræns objektets hastighed til dens
   maksimumhastighed

position = position + hastighed + Δt

vinkel = vinkel + vinkelhastighed + Δt
```

8.2.4 Player

Player klassen repræsenterer en spiller. En spiller har et unikt id som bruges til at identificere spilleren i netværksbeskeder og owner feltet i `Object`. `Player`

indeholder også spillerens navn og score. En spiller har et skib, som think metoden sørger for at genoprette hvis spilleren mister det, som er illustreret i liste 8.8.

Listing 8.8: Pseudo-kode for think i Player

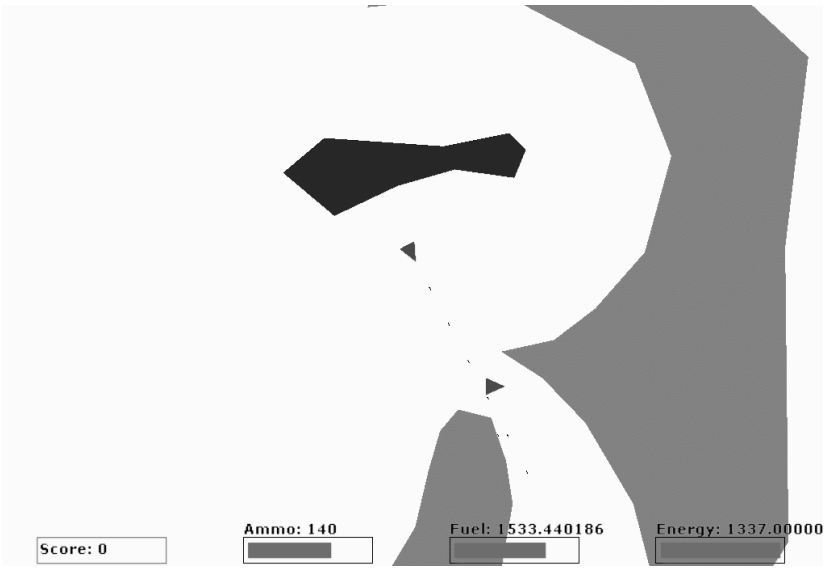
```
hvis spilleren ikke har noget skib
   hvis der er gået nok tid siden spilleren
      døde
      lav skib
      tilføj skib til bane
      sæt skibets ejer til spilleren
```

LocalPlayer

`LocalPlayer` nedarver fra `Player` og overskriver dens think metode. `LocalPlayer` eksisterer kun på klienten i en enkelt instans, nemlig den som repræsenterer klientens spiller. I `LocalPlayer`s think metode læses input fra brugeren, og skibet manipuleres afhængigt af input. Dette ses i liste 8.13. Den egentlige styring af spillerens rumskib håndteres passivt af den grund, at der så ikke skal tages specielt hensyn til spillerens skib nogle steder. `LocalPlayer` instansen gemmes i `GameControllerens` map af spillere på lige fod med de `Player` objekter som repræsenterer de fremmede spillere.

Listing 8.9: Pseudo-kode for think i LocalPlayer

```
kald think i Player

hvis spilleren ikke har et skib
   returner

hvis fremad knappen er nede
   accelerer spillerens skib
   brug lidt af skibets brændstof

hvis drej til højre knappen er nede
   sæt skibets vinkelhastighed således at det
      drejer til højre

hvis drej til venstre knappen er nede
   sæt skibets vinkelhastighed således at det
      drejer til venstre

hvis skydeknappen er nede
   bed skibet om at skyde

hvis selvmordsknappen er nede
   dræb skibet
```



Figur 8.1: Screenshot af Blaster

8.3 Netværk

I det følgende vil implementeringen af spillets netværk blive beskrevet. Beskeder i spillet vil først blive gennemgået, og derefter følger afsnit om den grundliggende netværkshåndtering i spillet.

Som udgangspunkt, skal klient og server opfylde kravspecifikationen. Dvs at det skal være muligt at kommunikere over UDP såvel som TCP. Netværket er implementeret vha. netværks-biblioteket SDL_net [Atk02]. SDL_net er dels valgt fordi projektet i forvejen benytter sig af SDL, og dels fordi SDL_net er et simpelt og let bibliotek.

Server og klient deler en række krav i kravspecifikationen:

- De skal begge kunne sende og modtage data via UDP
- De skal begge kunne sende og modtage data via TCP
- De skal begge kunne oprette og nedlægge TCP-forbindelser

Ydermere skal de begge kunne sende og modtage data af samme karakter. Det kan f.eks. være information om spilleres placering, spilleres energi osv.

Det lægger op til at server og klient deler en relativ stor del af netværks-koden. Dette er implementeret ved at lade server, såvel som klient, være klasser der nedarver fra en fælles superklasse `NetworkController`.

Der bruges forskellige protokoller med hver deres tilgang til at sende data. UDP er datagram (pakke) baseret, og TCP er socket-baseret[Ros01]. Det betyder at der skal implementeres to forskellige måder at sende beskeder hen over netværket på. For at lette behandlingen af beskeder er hver besked i spillet implementeret som en klasse.

8.3.1 Beskeder

Da alle beskeder deler visse funktionaliteter, såsom at de skal kunne afsendes, nedarver alle besked-klasser fra en fælles superklasse `NetMessage`. Beskeder i spillet oprettes via et såkaldt "Pluggable Factory"
citeplug[Cul99]. Foredelen ved et Pluggable Factory fremfor et normalt Factory er, at hvor et normalt Factory kan producere et fastlagt antal objekter, kan et Pluggable Factory producere et variabelt antal objekter der bliver fastsat compile-time.

Skulle beskeder i spillet være implementeret som et normalt Factory ville det i grove træk se ud som vist i kodeliste 8.10. Hver besked har en unik id, i dette tilfælde et tal. Når beskedFactory bliver kaldt med et tal, finder den frem til den tilhørende besked-klasse, og returnerer en ny instans af denne. Antallet af mulige besked-klasser er fast i metoden, så hvis man senere ønsker at udvide spillet med endnu en klasse, vil man skulle udvide denne metode.

Pluggable Factory

I et PluggableFactory, benytter man sig af et map over "Maker"objekter. Et Maker-objekt er et simpelt Factory, der kan producere et objekt af én bestemt type. Hver af disse Maker-objekter er i map'et associeret med en unik id. Når man ønsker et objekt af en bestemt type, slår man op i mappet og finder et Maker-objekt med den tilhørende id. Derefter bruger man Maker-objektet, til at producere det konkrete besked-objekt. Dette er vist i kodeliste 8.11. Maker-objektet kan virke som et unødigt mellemled, men da map'et sammenholder id'er med konkrete objekter, er det nødvendigt at indføre et led. Hvis man i stedet havde lagt besked-objekter i map'et ville man altid få returneret det samme objekt. Da der godt kan optræde flere beskeder af samme type ad gangen i spillet er dette uønsket.

Listing 8.10: Et Factory

```
beskedFactory ( id )

   hvis(id == 1)

      returner nyt besked1-objekt;


   ellers hvis ( id == 2)

      returner nyt besked2-objekt;

   .

   .

   .
```

Listing 8.11: Et PluggableFactory

```
pluggableBeskedFactory ( id )

   makerObjekt = beskedMap( id )

   returner makerObjekt.lavObjekt()
```

NetMessage

Alle beskeder nedarver fra klassen `NetMessage`. `NetMessage` indeholder en række funktioner alle beskeder har til fælles.

`getSize` Returnere størrelsen af pakken i bytes. Dette er f.eks. nødvendigt for at kunne bestemme størrelsen af pakken når beskeden sendes over UDP.

`serializeTo` Tager beskeden og skriver den til buffer i et format som beskeden senere kan gendannes fra. Dette bruges når beskeden skal sendes over netværket.

`serializeFrom` Læser en given mængde data fra en buffer, og indlæser en besked fra denne. Dette bruges når en besked modtages over netværket.

`think` Denne abstrakte metode bliver kaldt på subklasser af `NetMessage` når beskeden er nået frem. Think-metoden er "hjernen" i beskeden.

Derudover indeholder `NetMessage` 4 felter:

to Modtager af beskeden.

from Afsender af beskeden.

type Et tal der angiver hvilken type besked, der er tale om.

time Tidspunktet hvor hændelsen fandt sted.

PlayerKilled

`PlayerKilled` beskeden nedarver fra `SingleIntValuedMessage`, som nedarver fra `NetMessage`. I forhold til `NetMessage` indeholder `PlayerKilled` et spiller id.

Beskeden sendes fra serveren til alle klienter når en spiller dør. Beskeden sletter den relevante spillers skib, som vist i liste 8.12.

Listing 8.12: Pseudo-kode for think i PlayerKilled

```
find spilleren med det givne id


hvis spilleren findes

   dræb spillerens skib
```

ShipUpdate

`ShipUpdate` beskeden opdaterer informationen omkring et skib. Dette inkluderer dets placering og orientering i spillet, samt mængden af ammunition og brændstof skibet har tilbage. Beskeden kan sendes både fra en klient eller en server. Hvis den kommer fra en klient, er det klientens eget skib som skal opdateres på serveren. Hvis den kommer fra serveren, er det oftest andre spilleres skibe der skal opdateres, men ved kollision har serveren brug for at fortælle klienten at den skal ændre retning.

`ShipUpdate` er ikke implementeret i skrivende stund. Det ville være trivielt at implementere, men indtil det er muligt at modtage beskeder ansås det ikke som vigtigt at implementere flere beskeder. Det ansås dog som vigtigt at beskrive klassen, da den ville være en central del af netværksfunktionaliteten når/hvis den blev implementeret.

Listing 8.13: Pseudo-kode for think i LocalPlayer

```
find spilleren med det givne spillerid


hvis spilleren ikke findes
   returner


skib = spillerens skib


overskriv skibets position med det fra
   beskeden
overskriv skibets hastighed med det fra
   beskeden
overskriv skibets acceleration med det fra
   beskeden
overskriv skibets orientering med det fra
   beskeden
```

8.3.2 NetworkController

`NetworkController` håndterer oprettelse og nedlæggelse af alle forbindelser imellem klient og server. Serveren har brug for at kunne kontrollere flere forbindelser til flere forskellige klienter. `NetworkController` arbejder derfor med en liste af forbindelser. Klienten har kun brug for en en enkelt udadgående for-

bindelse til serveren, men da der ikke er noget i vejen for blot at bruge en liste med et en enkelt forbindelse i, ændrer dette ikke på implementeringen.

TCP er en forbindelses-orienteret, socketbaseret protokol. UDP er en forbindelsesløs datagrambaseret protokol. Man kan derfor ikke tale om en egentligt UDP-forbindelse. Det er dog vigtigt for server såvel som klient at holde styr på om man har mistet forbindelsen med den anden ende. Dette er håndteret ved at lave en forbindelses-klasse(`Connection`). Klassen indeholder en TCP-forbindelse(socket), et forbindelses-id (`ConnectionID`) samt en IP-adresse. IP-adressen bruges når man skal sendes data via UDP, og TCP-forbindelsen bruges, når der skal sendes data via TCP. Hvis TCP-forbindelsen på noget tidspunkt fejler, eller bliver nedlagt, slettes forbindelses-objektet. På denne måde bruges TCP-forbindelsen til at kompensere for manglen på forbindelses-orientering i UDP.

Ind- og udgående data er begge implementeret som køer. Når man ønsker at sende f.eks. UDP-data, skubbes den først på den udgående UDP-kø, der herefter tømmes af `NetworkController`. Der findes ligeledes en kø for udadgående TCP-data. Indgående UDP- og TCP-data håndteres begge via den samme kø, da protokollen en besked leveres med ikke har nogen betydning for spillet.

Oprettelse og nedlæggelse af forbindelser

Metoderne `addConnection` og `removeConnection` bruges til at oprette og nedlægge forbindelser. Når en forbindelse oprettes, indsættes den i listen af forbindelser. Når den nedlægges, fjernes den fra listen. Pseudo-koden for de to metoder kan ses i listing 8.14 og 8.15.

Listing 8.14: addConnection-metoden

```
opret TCP-forbindelse med SDL_net

opret et nyt Connection-objekt

indsæt objektet i forbindelses-listen
```

Listing 8.15: removeConnection-metoden

```
find forbindelsen i forbindelses-listen

nedlæg TCP-socket med SDL_net

slet forbindelses-objektet fra forbindelses-listen
```

Køer i NetworkController

`NetworkController` indeholder 3 køer:

`messageInQueue` der er en kø af indkommende beskeder.

`UDPOutQueue` er en kø af udgående beskeder der skal sendes via UDP.

`TCPOutQueue` er en kø af udgående beskeder der skal sendes via TCP.

Når man vil sende beskeder foregår det v.ha. metoderne `TCPSendMessage` og `UDPSendMessage`. Disse metoder tager en besked som argument, og putter den på den respektive kø. `NetworkController` indeholder metoden `readMessages`, der kaldes med jævne mellemrum. Denne metode gennemløber alle forbindelser i forbindelses-listen, og indlæser evt indkommende data. Denne data laves v.ha. `MessageMakerBase` om til beskeder, og puttes på den indkommende besked-kø. Til sidst bliver `think` metoden på alle beskeder i besked-køen kaldt, således at alle beskeder bliver udført.

Pseudo-koden for metoden `flush`, der sender de udgående køer findes i kode-liste 8.16. Pseudo-koden for metoden `readMessages` findes i kode-liste 8.17.

Listing 8.16: flush-metoden fra NetworkController

```
sålænge der er beskeder i TCP-køen

   serialiser beskeden

   send beskeden vha SDL_net

   fjern beskeden fra TCP-køen


sålænge der er beskeder i UDP-køen

   serialiser beskeden

   send beskeden vha SDL_net

   fjern beskeden fra UDP-køen
```

Listing 8.17: readMessages-metoden fra NetworkController

```
for hver forbindelse i forbindelses-listen

   sålænge der er data på TCP-forbindelsen

       opret ny besked ud fra data

       skub beskeden på besked-køen


sålænge der er data på UDP-forbindelsen

   opret ny besked ud fra data

   skub beskeden på besked-køen
```

```
for hver besked i besked-køen

   kald think-metoden på beskeden

   fjern beskeden fra køen
```

Afsnit 9

Afprøvning af spillet

I det følgende vil afprøvning af spillet blive gennemgået. Først vil den valgte test-strategi blive beskrevet, herefter følger en række eksempler på de udførte tests. Samtlige tests kan findes i bilag B.1 og E.

9.1 Afprøvnings strategi

Når spillet skal testes tages der udgangspunkt i kravspecifikationen, gennemgået i afsnit . Der foretages eksterne, og visuelle tests. De eksterne tests udføres v.ha. en række test-klasser. Disse klasser opstiller et senario, og bruger C++' `assert`-funktion til at undersøge om testen fejler. Kildekoden til de implementerede test-klasser kan findes i bilag E.

De visuelle tests er udført som en række trin. For hvert trin opstilles handling, det forventede resultat, samt det egentlige resultat.

Ud fra kravspecifikationen skal følgende overordnede ting afprøves:

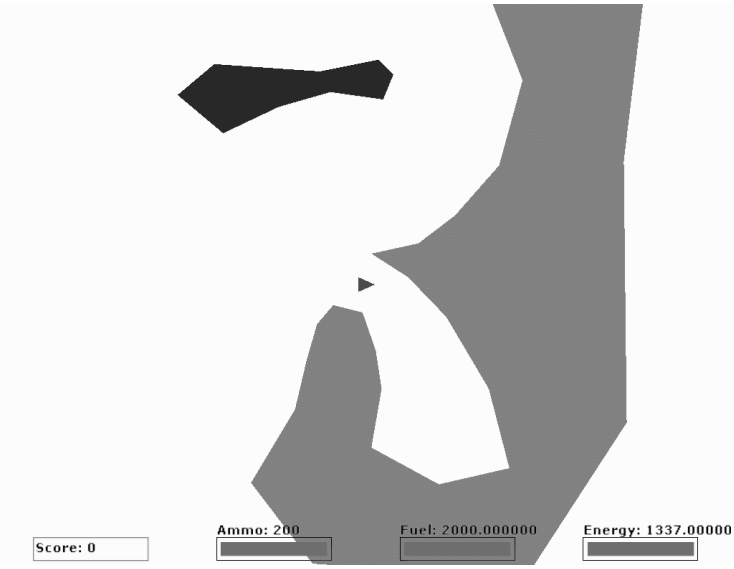
- Kommunikation imellem fra/til server/klient via UDP og TCP
- Fjernelse og tilføjelse af klienter fra/til spil på server såvel som klient
- Serverens autoritære rolle
- Indlæsning af baner på server og klient
- Kollisioner på serveren
- Vedligeholdelse af information om spillere på server såvel som klient
- Visualisering af spillet på klienten
- Brugerinteraktion på klienten

Spillet var ikke fuldt implementeret da de nedenstående tests blev udført. Det betød at visse tests har måtte udelades. Det drejede sig om afprøvning af serverens autoritære rolle, da netværks-beskeder ikke fungerede, og server/klient derfor ikke kunne kommunikere.

9.1.1 Visuel afprøvning af spillet

- Start af spillet
- Bevægelse af skibet
- Rotation af skibet
- Kollision imellem skib og bane
- Affyring af skud
- Udtømning af brændstof

De 2 første scenarier, "start af spillet" og "rotation af skibet" er vist her. Samtlige scenarier kan findes i bilag B.1.1.

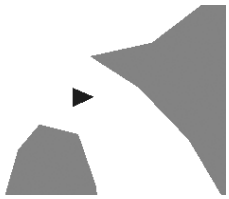


Figur 9.1: Spil ved start

Start af spilet

Nr.	**Navn**	
1	Start af spilet	
Beskrivelse		
Spillet skal kunne starte		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Start spillet	Spillet starter og viser en bane, et skib, samt en række status-bjælker i bunden af skærmen.	Spillet starter og der ses en bane med et trekantet skib i midten. I bunden findes 3 status-bjælker, og en point-angivning.
Afprøvning bestået		Ja, se figur 9.1

Figur 9.2: Lige før bevægelse



Figur 9.3: Lige efter bevægelse

Bevæg skibet frem

Nr.	**Navn**	
2	Bevæg skibet frem	
Beskrivelse		
Når der trykkes på fremdrifts-knappen (pil op), skal skibet flyve fremad, såfremt der er mere brændstof		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Se test 1	-	-
Tryk på fremdrifts-knappen	Skibet accelereres	Skibet accelererer, brændstofmåleren i bunden af skærmen falder.
Afprøvning bestået		Ja, se figur 9.2 og 9.2

Del III

Editor

Afsnit 10

Kravspecifikation

På baggrund af analysen i afsnit 5, opstilles følgende kravspecifikation for baneeditoren:

- Editoren skal være i stand til at tegne polygoner bestående af tre punkter og kanter mellem disse.
- Polygoner skal indgå i et bane
- Polygoner skal kunne flyttes rundt i denne bane
- Polygoner skal kunne fjernes fra en bane
- Farven på en polygon skal kunne ændres
- Hvis et nyt punkt kommer indenfor en vis minimumsafstand af et ekstisterende punkt, antages de at være et punkt og de "smelte" sammen.
- Det skal være muligt at flytte punkter og kanter i en polygon.
- Editoren skal være i stand til at gemme en bane i et format, som spillet er i stand til at læse og senere hente ind i editoren igen.
- En bane skal have en størrelse.
- En bane skal have et navn.
- En bane har tyngdekraft.
- Det skal specificeres om en kant er en platform.

Afsnit 11

Designovervejelser

Følgende afsnit beskriver overordnet opbygningen og ideerne bag opbygningen af editoren. Desuden gennemgås brugergrænsefladen for editoren.

11.1 Generelle design ideer

I designet af en større grafisk applikation i et objekt orienteret sprog som Java, er det nærliggende at benytte sig af Model/View/Controller (MVC) mønstret [Gam94],[Rob03]. Dette passer godt ind i Java Swing da det jo netop er bygget op omkring MVC mønstret.

Programmet består derfor af tre dele. En model, et view og en controller-del. Model-delen består af selve banerepræsentationen dvs. polygoner, punkter og kanter. Controller-delen består af dele, der manipulerer modellen, dvs. klasser der kan flytte og danne polygoner, kanter og punkter. Endelig består view delen af klasser, der kan tegne modellen og håndtere brugerens interaktion med programmet gennem controller delen.

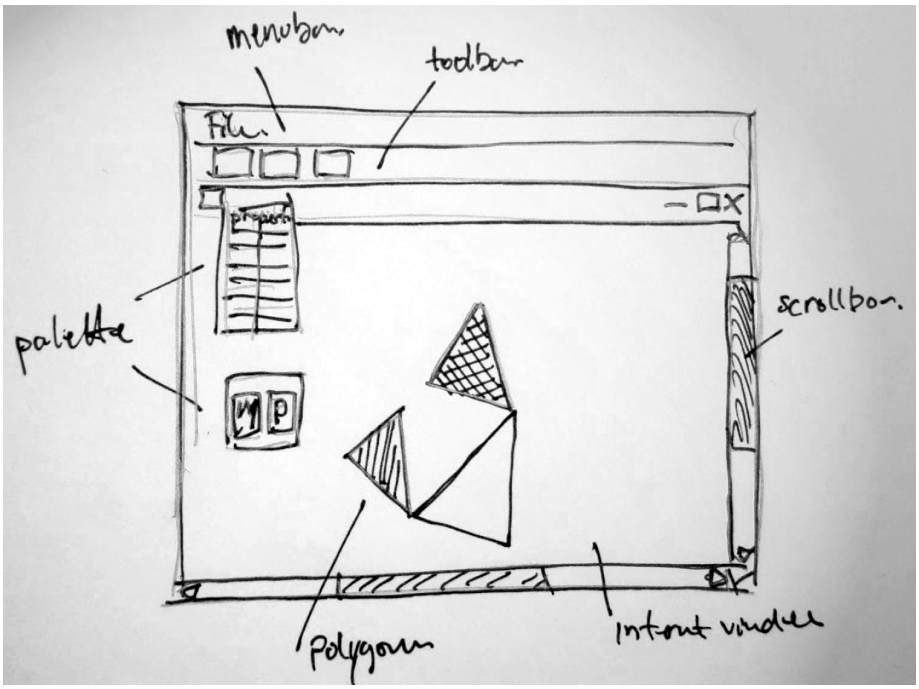
Udover at være bygget over MVC mønstret, der i forvejen fordrer nem udvidelse af programmet, er der i designet lagt vægt på at gøre programmet nemt at udvide med f.eks. mere avancerede polygonmanipulationer. Det kunne f.eks. være rotation af polygoner, teksturer på polygoner eller sågar visuel feedback på operationer.

11.2 Brugergrænseflade

I designet af brugergrænsefladen er der *ikke* benyttet metoder til at sikre at applikation tilbyder de faciliteter, der kræves på nemmest og mest overskuelig vis. Der er derfor heller ikke udført undersøgelser af om brugergrænsefladen opfylder en brugers behov.

Programmet er bygget op omkring et Multiple Document Interface (MDI). Hvorvidt et MDI er godt eller dårligt diskuteres jævnligt. Men da det er den eneste måde at behandle flere dokumenter i et Java-program på en nem og overskuelig måde, blev denne metode valgt. Java's implementation af MDI er dog ikke helt uproblematisk på platforme, hvor MDI ikke eksisterer f.eks. Mac OS X. Men

dette er for langt fra projektets egentlige fokus til at der er blevet overevejet alternative måder at løse problemet på.



Figur 11.1: Mockup af editor

Programmet indeholder to paletter. Én til at skifte mellem to mulige funktionalitets tilstande. At lave polygoner vha af punkter og at vælge polygoner, kanter og punkter samt at flytte disse. En anden palette indeholder en tabel over alle de mulige egenskaber for det valgte polygon, kant eller for hele banen.

Alle baner åbnes i et nyt vindue der kan lukkes, maksimeres og minimeres. Titlen på vinduet er identisk med banens titel.

Programmets værktøjsbjælke indeholder knapper til at lave en ny bane, hente en bane, gemme en gane og eksportere en bane til formatet, der bruges i selve spillet. Det samme gælder menubjælken.

11.3 Overordnet design

Det er forsøgt at overføre kravene fra kravspecifikationen og brugergrænsefladens begrænsinger til et design efter MVC mønstret. Det er dog ikke alle steder at opdelingen er lykkedes lige godt, eller at opdelingen har været hensigtsmæssig. MVC mønstret er ikke svaret på alle bønner, og det kan i visse situationer forvirre mere end det gavner, særligt i simple programmer som editoren må antages at være. En inddeling af områderne efter MVC kan ses i figur 11.2 på side 53

11.3.1 Model

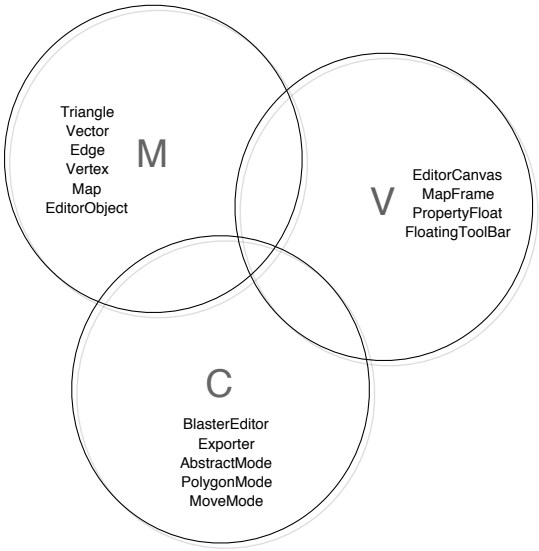
Banen, polygoner, kanter og punkter kan umiddelbart anskues som klasser. Deres ansvar er at indeholde data om objekters placering i banen og deres egenskaber samt banens egenskaber. Da der er en del fælles funktioner på objekterne vil

det være naturligt at lade dem alle arve fra en fælles superklasse. Dette gør det muligt at behandle klasserne via en fælles grænseflade. Den fælles grænseflade indholder metoder til at sætte egenskaber, hente egenskaber og flytte samt vælge objekter.

11.3.2 View og Controller

Hver bane skal tegnes i et vindue, der er en del af selve applikations vinduet. Der er derfor brug for en del der kan tegne banen samt modtage input fra brugeren til at manipulere banen. Denne skal kunne eksistere i flere instanser i selve applikationen.

For at opnå en bedre fordeling af ansvaret deles dette i to hovedgrupper. Selve view delen, der sørger for at tegne banen og en input og manipulations del kaldet controller. Input-delen sørger for at delegere input til manipulationsdelen. Manipulations-delen sørger for at kalde metoder på banens objekter for at manipulere objekterne.



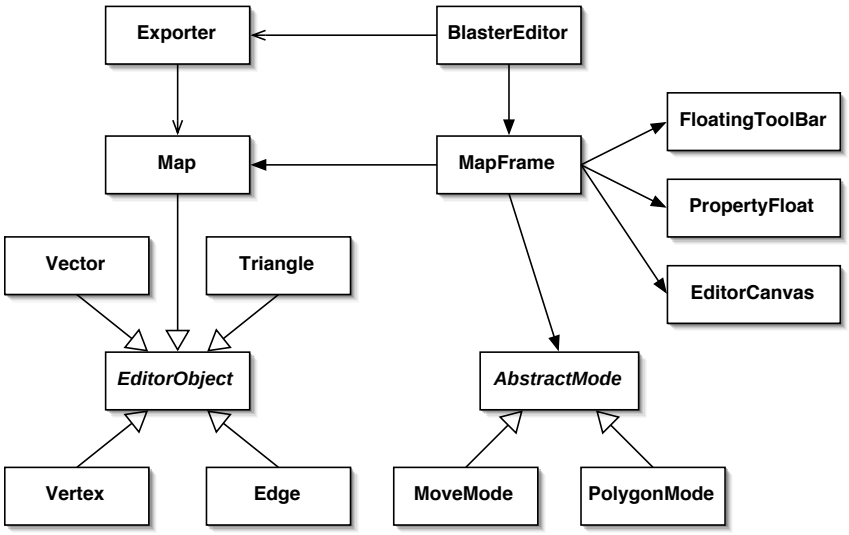
Figur 11.2: Klasserne indelt i MVC

Selve brugergrænsefladen består af hovedklassen `BlasterEditor` der indeholder hvert vindue, hvor banen konstrueres. Hver vindue repræsenteres af `MapFrame`. BlasterEditor indeholder desuden de to paletter `FloatingToolBar` og `PropertyFloat`, der begge er implementeret som singletons, da de skal kunne bruges fra flere forskellige klasser, og for begges vedkommende kun bør findes en instans.

Manipulationer håndteres af `AbstractMode` og klasser der arver fra denne henholdsvis `MoveMode` og `PolygonMode`. Disse modtager input BlasterEditor der lytter på selve events fra brugeren og sørger for at udføre manipulationen af banen.

Den sidste del er eksport og lagring af banefiler. Dette hører til i controller-delen og henter blot informationer fra et baneobjekt for at skrive relevante informationer til filsystemet.

En oversigt over klasserne og deres sammenhæng kan ses på figur A.8.



Figur 11.3: Klasserne i editoren og deres sammenhæng

Afsnit 12

Implementering af editor

I de følgende afsnit vil implementeringen af editoren blive gennemgået. Først vil opdelingen af editoren blive gennemgået, og derefter vil centrale dele og mønstre blive gennemgået. Mindre interessante dele som f.eks. definition af brugergrænsefladen er undladt, da det stort set kun består af API kald.

Editoren er som før beskrevet opbygget efter MVC-mønsteret. Da dette lægger op til en klar adskillelse imellem data og fremvisningen af denne er editoren opdelt i 3 pakker.

- UI (User Interface)
- Model
- Export

Model-pakken repræsenterer data i editoren. Klasserne i ui-pakken indeholder alt, hvad der har med visning af data, samt brugerinteraktion at gøre. Model er pakken som navnet antyder modellen, og ui-pakken er både controller og view. `Export` pakken er et view af dataen i XML.

Jfr. figur 12.1 på side 56. `BlasterEditor` er et `JFrame`, der indeholder min 3 vinduer indeni sig. Derudover er der en normal menubjælke, der har knapper til at indlæse, gemme og eksportere banen. De tre vinduer er:

1. Et view af banen (`MapFrame`)
2. Knapper til at skifte tilstand (`FloatingToolBar`)
3. En liste over egenskaber (`PropertyFloat`)

Først et view af selve banen. Vinduet er implementeret i `MapFrame` der nedarver fra `JInternalFrame`. Indholdet af vinduet bliver optegnet af `EditorCanvas` der er implementeret som et `JPanel`.

Editoren kan skifte imellem forskellige tilstande(Modes). Hver af disse tilstande er implementeret som en klasse. I den nuværende implementering af editoren er der 2 tilstande: én til at tegne polygoner samt én til at flytte elementer i banen. Når editoren er i polygon-tilstand kan brugeren tegne polygoner, og i flytnings-tilstand kan hele polygoner flyttes.

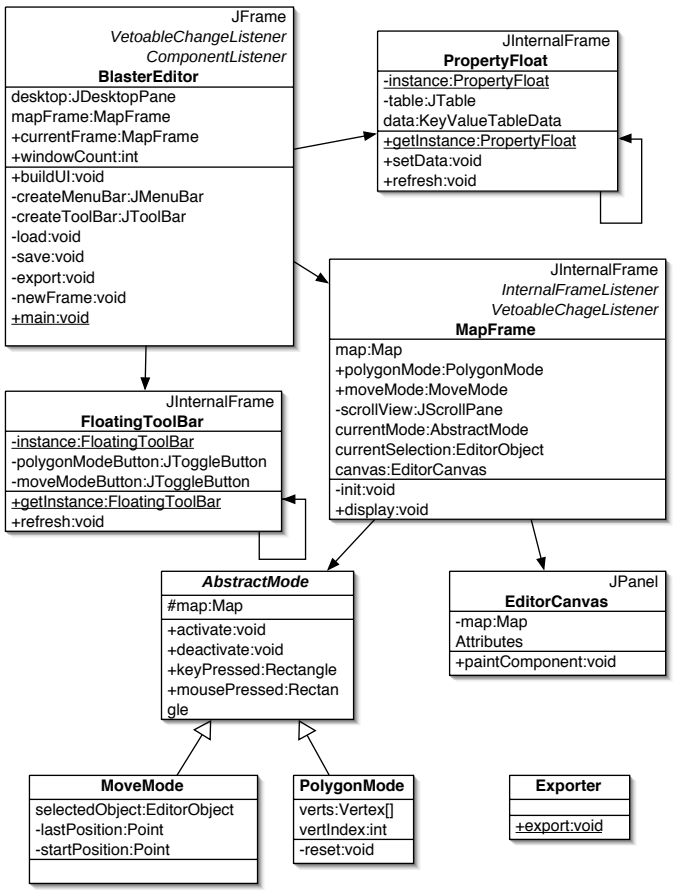
`Modelpakken` arbejder med model-dataen i editoren. Editoren arbejder med kanter, polygoner og punkter. Disse bliver repræsenteret af hver sin klasse:

- Kanter repræsenteret ved klasssen `Edge`
- Tresidede polygoner repræsenteret ved klasssen `Triangle`
- Punkter repræsenteret ved klasssen `Vertex`

Klassen `Map` repræsenterer banen som en helhed. Den indeholder lister af de punkter og trekanter, der indgår i banen, samt informationer om selve banen, så som højde og bredde. Samtlige klasser i model-pakken nedarver fra EditorObject (EditorObject selv undtaget), af hensyn til at have et fælles interface til at håndtere egenskaber for elementerne.

Export-pakken indeholder en enkelt klasse, exporter. `exporter` bruges til at eksportere en bane ud i et XML-format, der senere kan indlæses af spillet.

12.1 Centrale dele

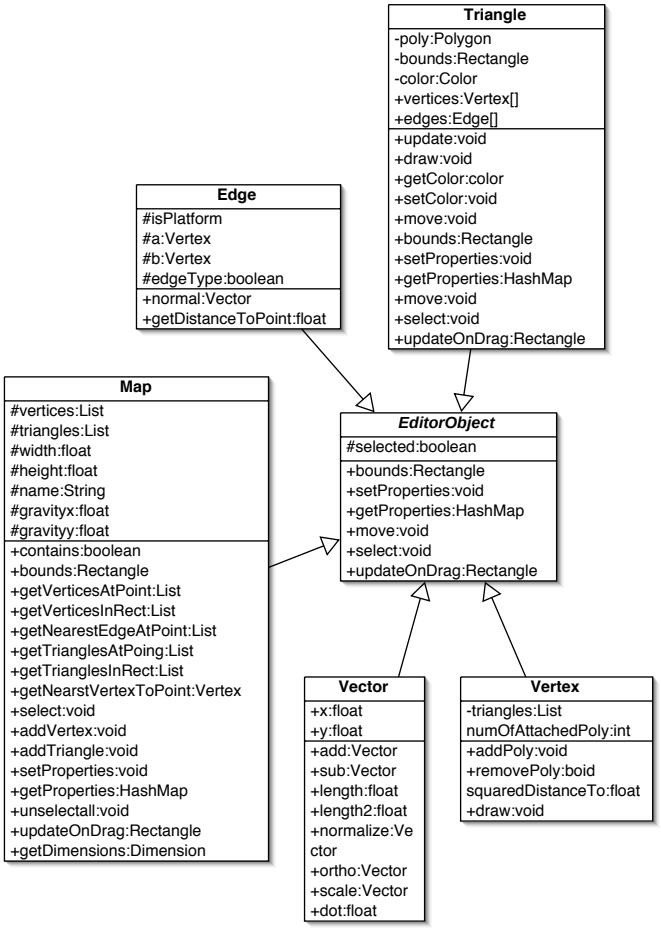


Figur 12.1: Klasserne i ui-pakken

12.1.1 Baneelementer

Banen består af kanter, punkter og polygoner. Disse objekter har en række ting til fælles.

- De kan flyttes



Figur 12.2: Klasserne i model-pakken

- De har tilknyttet egenskaber så som farve og størrelse
- De kan vælges

Det er derfor oplagt at lade den nedarve fra en fælles baseklasse. `EditorObject` er en abstrakt klasse der indeholder den funktionalitet som baneelementerne deler. Jfr. figur 12.2 på side 57

Metoder bane-elementer deler:

`setProperties` og `getProperties` metoderne bruges til at få adgang til en liste af egenskaber tilknyttet objektet.

`move` metoden flytter objektet til et givet koordinatsæt.

`select` metoden sætte objektet til at være valgt eller ikke valgt.

`bounds` metoden returnerer et rektangel, der omslutter objektet.

`updateOnDrag` returnerer et rektangel, der angiver hvilket område på skærmen, der skal opdateres efter at en objekt er blevet flyttet.

Map nedarver også fra `EditorObject`. Dette er gjort fordi `Map` indeholder en række egenskaber som navn og størrelse. `move` og `updateOnDrag` metoderne fra `EditorObject` er overskrevet med tomme metoder, da brugeren ikke skal kunne flytte et map.

12.1.2 Præsentation af modellen

Brugergrænseflade

`EditorCanvas` sørger for den egentlige tegning af banen ved at kalde `draw` metoden i `Map`. Denne metode beder hvert element i banen om at tegne sig selv til EditorCanvas' kontekst.

Eksportering

Klassen `Exporter` kan eksportere banen fra editoren ud i et XML-format. Selve eksporteringen sker v.h.a. Java-pakken `ElectricXML`. Den er valgt da den er meget let at arbejde med. Hvis ikke programmet skulle understøtte mange platforme kunne man med fordel have brugt Javas egen XML parser, der er med i alle versioner efter 1.4.0. Der er dog endnu ingen 1.4 version til Mac OS X.

Den resulterende XML-fil indeholder data fra banen, struktureret på stort set samme måde som den er struktureret i editoren:

- En liste over punkter, hvor hvert punkt tildeles et fortløbende nummer.
- En liste over polygoner, hvor hver polygon tildeles et fortløbende nummer.
- For hver polygon: en liste over kanter
- For hver kant: referencer til de to punkter som kanten er dannet af. Jfr figur 12.1

Listing 12.1: Pseudo-kode for eksportering af Map

```
tilføj ny map-rod

tilføj ny punkt-rod til map-rod
for hvert punkt i map
   tilføj punkt til punkt-rod

tilføj ny polygons-rod
for hver polygon i map
   tilføj ny polygon-rod
   for hver kant i polygon
      find id for start/slutpunkt
      tilføj punkt til polygon-rod
eksporter fil
```

En eksporteret XML fil for en bane med to polygoner kan se således ud:

```
<?xml version='1.0' encoding='UTF-8'?>
<map width='1000.0' height='1000.0'
     gravityx='0.0' gravityy='0.0' name=''\>
  <points size='6'>
    <point x='47.0' y='-87.0' id='0'/>
    <point x='109.0' y='-45.0' id='1'/>
    <point x='52.0' y='-33.0' id='2'/>
    <point x='126.0' y='-46.0' id='3'/>
```

```
    <point x='128.0' y='-102.0' id='4'/>
    <point x='63.0' y='-100.0' id='5'/>
  </points>
  <polygons size='2'>
    <polygon id='0'>
      <color r='1.0' g='1.0' b='1.0'/>
      <edge p1='0' platform='false' id='0'/>
      <edge p1='2' platform='false' id='1'/>
      <edge p1='1' platform='false' id='2'/>
    </polygon>
    <polygon id='1'>
      <color r='1.0' g='1.0' b='1.0'/>
      <edge p1='3' platform='false' id='0'/>
      <edge p1='4' platform='false' id='1'/>
      <edge p1='5' platform='false' id='2'/>
    </polygon>
  </polygons>
</map>
```

12.1.3 Gem bane fra editor

Da editoren har brug for flere informationer om banen end spillet, blev det besluttet at lave to formater. Spillets eget format er en simpel serialisering af Java objektet `Map`. Der er ikke taget stilling til hvilke felter der skal serialiseres. Der er bare benyttet Java standard serialisering ved at implementere Serializable interfacet. Dette gør det meget nemt både at gemme og hente banen ind igen, men svært at bruge banen i andre programmer. Men man slipper for at skrive parsere til et filformat.

12.1.4 Tilstande

Muligheden for at skifte tilstande i editoren er implementeret som et Strategy mønster. Der er dermed mulighed, for at implementere vilkårligt mange tilstande i editoren.

Jfr. figur 12.1 på side 56. Alle tilstands-objekter nedarver fra den abstrakte klasse `AbstractMode`. `AbstractMode` indeholder abstrakte metoder til at håndtere input fra brugeren, som det konkrete objekt skal implementere. `MapFrame` indeholder en reference til det nuværende tilstands-objekt, af typen `AbstractMode`.

MapFrame bruger sit AbstractMode til at håndtere input fra brugeren. Dermed bestemmer den aktuelle tilstands-objekt, hvordan editoren reagerer på input.

Polygon-tilstand

Polygon-tilstand(PolygonMode) bruges til at tegne nye polygoner.

I polygon-tilstand reagere editoren på følgende måde:

- Hvis brugeren klikker på et vilkårligt sted på banen, tilføjes der et nyt punkt på det pågældende sted.

- Når det tredje punkt tilføjes, oprettes der en ny trekant bestående af disse 3 punkter.
- Hvis et nyt punkt sættes inden for en minimumsafstand af et eksisterende punkt samles de to til et punkt

For at tilføje et punkt afsættes tre punkter på en `MapFrame`. For hver gang et punkt indsættes i den interne liste af punkter undersøges det om der er kommet tre punkter siden sidste polygon blev dannet. Hvis dette er tilfældet dannes en ny polygon. Alt dette forgår i `PolygonMode` klassen. Dette er illustreret i den følgende pseudo kode:

Listing 12.2: Pseudo-kode for tilføjelse af punkt

```
tilføj punkt til Map's punkt-liste

hvis antal i Map's punkt-liste = 3
   trekant = lav trekant fra punktliste
   tilføj trekant til map
```

Hvis et nyt punkt ved tilføjelse af punkt til liste viser sig at ligge inden for en bestemt minimums afstand fra et eksisterende punkt skal disse samles. Dette sker i `Map`'s `addVertex metode`, men samme kode. Dette er illustreret i følgende pseudo kode (Liste 12.3):

Listing 12.3: Pseudo-kode for samling af to punkter

```
punkt = find nærmeste-punkt

hvis punkt == tom
   punkt = lav nyt punkt
tilføj punkt til punkt-liste

returner punkt
```

MoveMode

Flytningstilstand (MoveMode), bruges til at flytte rundt på bane-elementer.

I flytnings-tilstand reagere editoren på følgende måde:

- Hvis brugeren klikker på et bane-element (punkt, kant polygon) bliver dette markeret som valgt
- Hvis brugeren vælger et bane-element og holder museknappen nede, flyttes elementet sammen med musen.
- Hvis to punkter (ved flytning af punkter) kommer inden for en minimums-afstand, samles disse til et punkt.
- Hvis brugeren trykker på `delete`-knappen på sit keyboard, slettes det valgte bane-element.

For at vælge et objekt i et Map kaldes `select` med et punkt som argument. Denne metode undersøger først om der er nogen punkter inden for en bestemt

radius. Hvis ikke undersøges for kanter og hvis ingen kanter findes; da undersøges for polygoner. Hver af metoderne "find nærmeste ..." finder nærmeste punkt, kant eller polygon inden for en begrænset radius omkring et punkt. Findes ingenting returneres selve Map. Det kan ses i pseudokode herunder (Liste 12.4):

Listing 12.4: Pseudo-kode for valg af bane-element

```
BaneElement element

element = find nærmeste punkt
hvis element ikke tom
   returner element

element = find nærmeste kant
hvis element ikke tom
   returner element

element = find nærmeste polygon
hvis element ikke tom
   returner element

returner Bane
```

Hver af disse metoder "find nærmeste punkt" "find nærmeste kant" og "find nærmeste polygon" findes i Map. Logikken i "find nærmeste punkt" med argumentet "punkt" går ca således:

Listing 12.5: Pseudo-kode til at finde nærmeste punkt

```
for alle punkter p i punkt-liste
   hvis p indeholder punkt
      returner p
   ellers
      returner tom
```

Metoden undersøger hvorvidt et punkt er indenfor en given radius. Resten af metoderne minder om ovenstående.

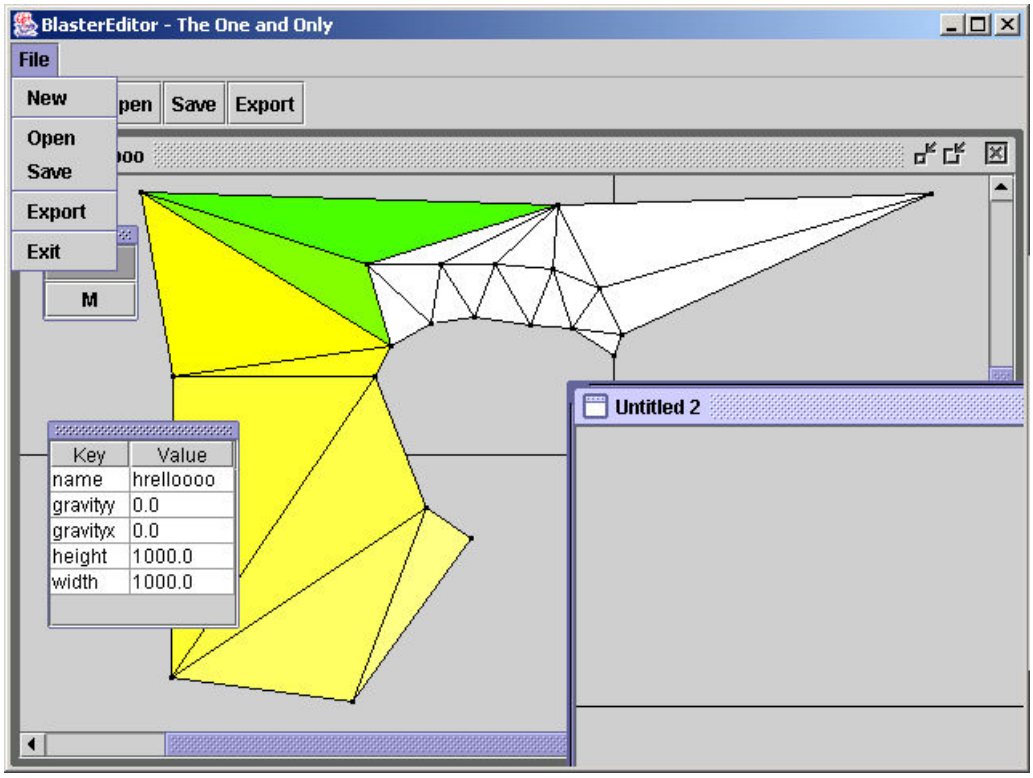
Hvis to punkter er tilstrækkelig tæt på hinanden skal de samles som i PolygonMode. Dettes gøres fra MoveMode's input håndtering. I pseudo kode ser det ud som følger:

Listing 12.6: Pseudo-kode for samling af to punkter

```
nærmestePunkt = find nærmeste-punkt

hvis nærmeste-punkt inden for margin
   for hvert polygon p i Map
      for hver punkt pk i p
         hvis pk = p
            pk = nærmeste-punkt
ellers
   fortsæt
```

12.2 Resultat

Editoren ser ud til at leve op til alle krav. Et billede af editoren kan ses på figur 12.3



Figur 12.3: Editor i fuld funktion, i baggrunden ses en bane med farvede polygoner, i forgrunden en tom bane

Afsnit 13

Afprøvning af editor

Følgende afsnit vil præsentere en strategi for afprøvning af editoren samt resultater af afprøvningen. Afprøvningen er udført som ekstern afprøvning. Yderligere resultater er at finde i bilag B.

13.1 Afprøvnings strategi

For at afprøve editoren tages udgangspunkt i kravspecifikationen i afsnit 10 side 50. Herfra er der opstillet afprøvnings scenarier for hvert krav.

Det har naturligvis ikke været muligt, ej heller ønskværdigt, at præsentere alle mulige scenarier. Hvorfor der kun er udvalgt nogle repræsentative scenarier til dette afsnit. Desuden har der p.g.a. manglende tid ikke været muligt at gennemføre alle scenarier. Ydermere er det enkelte scenario kun afprøvet på en af de understøttede platforme.

Afprøvningen af editor er delt op i 2 hovedgrupper. Brugerafprøvning og ekstern afprøvning af specifikke klasser og algoritmer.

Brugerafprøvningen sikrer at programmet opfører sig som brugeren forventer. Dette gøres ved at opstille nogle scenarier for de krav, der er stillet til programmet fra en brugers synspunkt. En brugers handling skridt for skridt opstilles i et skema, med forventet resultat og egentlige resultat for hvert skridt. Derved kan man danne sig et overblik over eventuelle mangler eller uhensigtsmæssigheder i forhold til ventet.

Den eksterne afprøvning kom aldrig videre end planlægningsfasen p.g.a. den førnævnte mangel på tid. Dette var tænkt udført for nogle få repræsentativt udvalgte klassers vedkommende, ved at opstille ækvivalensklasser for disse. Dette skulle afprøves v.ha. JTest's indbyggede understøttelse for "Design by Contract" [Mey02]. JTest er et værktøj til automatiseret afprøvning af Java programmer.

Desuden er der skrevet unit tests af nogle af klasserne. De er alle skrevet i forbindelse med udviklingen af projektet for løbende at kunne undersøge om klassen fungerede som tiltænkt. Uagtet hvilken stand resten af systemet befandt sig i.

Unit tests er udført v.ha. værktøjet JUnit der er et hjælpeværktøj til at udføre unit tests af java kode.

13.2 Brugerafprøvning af editor

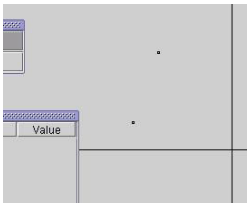
Der er kun medtaget to scenarier fra afprøvningen i denne rapport, resten kan findes i bilag B.2.1 på side 90. Scenarierne er følgende:

- Tre punkter danner polygon
- Vælg og flyt polygon
- Slet polygon fra bane
- Skift farve på polygon
- Sammensmeltning af to punkter
- Flytning af kant

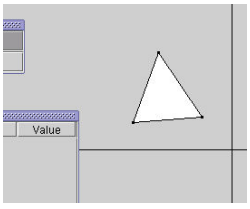
De to er udvalgte er "Tre punkter danner polygon" og "Skift farve på polygon" .

Nr.	**Navn**
1	Tre punkter danner polygon
Platform	MacOS X
Beskrivelse	
Ved at afsætte tre punkter efter hinanden i editoren dannes en udfyldt trekant umiddelbart efter det tredje punkt er afsat.	

Trin beskrivelse	**Forventet resultat**	**Resultat**
Åbn editoren	Editoren startet, klar til input	Editoren er klar til input
Lav en ny bane	Et nyt bane vindue i editoren	Bane vindue klar til input
Afsæt et punkt i banen	Banen indeholder et punkt og der hvor der blev trykket ses nu en lille firkant	Et lille punkt ses i bane vinduet
Afsæt endnu et punkt lidt nord vest for det første punkt	Endnu et punkt i editoren	En lille firkant til
Afsæt endnu et punkt øst for det andet punkt	En hvid trekant dannes med sorte kanter og endnu en lille firkant	Der dannes en trekant
Afprøvning bestået		Ja, se evt figur 13.1, 13.2



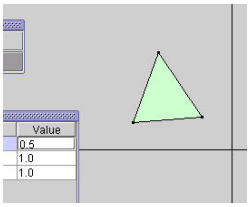
Figur 13.1: To punkter afsat



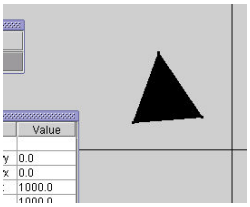
Figur 13.2: Trekant dannet

Nr.	**Navn**
4	Skift farve
Platform	MacOS X
Beskrivelse	
Ved at ændre i farve egenskaberne for en polygon skifter polygonet farve	

Trin beskrivelse	**Forventet resultat**	**Resultat**
Følg scenarie 1	-	-
Vælg polygonen og skift farve	Polygonen skifter farve	Polygonen bliver sort, men først efter at polygonen ikke længere er valgt. Måske burde farven der angiver at en polygon er valgt være en toning af polygonens farve
Afprøvning bestået		Ja, men måske bør der ses på hvordan en polygon vises som værende valgt, se evt figur 13.3, 13.4



Figur 13.3: Polygon valgt



Figur 13.4: Polygonen farvet

13.3 Unit test

Resultatet af unit test kørslen kan findes i bilag B.2.2.

13.3.1 Afprøvning af `Map`

I `Map` klassen afprøves metoderne: `getProperties` for at verificere at egenskaberne for map opdateres ved en setProperties og at de kommer ud igen i samme stand. `getVerticesInRect(Rectangle)` for at verificere at man ved at specificere et område i planet får returneret hvilke `Vertex` objekter der ligger inden for. `getTrianglesInRect(Rectangle)` for at undersøge om man får de `Triangles` tilbage der ligger i `Rectangle`. `select(Point)` undersøges for at se om der returneres `Vertex`, `Edge` og `Triangle` afhængig af `Point`.

Fælles for alle metoderne er at der forinden er lavet en bane med to polygoner, der ligger op ad hinanden. Selve koden kan findes i bilag E.2.

`getVerticesInRect(Rectangle` og `getTrianglesInRect(Rectangle)`

Et kald til getVerticesInRect kan resultere i 0, 1 eller flere returnerede objekter. Derfor konstrueres 3 kald til metoderne, et der bør give 0, et andet 1 og endelig et der skal returnere to objekter.

`select(Point)`

`select(Point)` returnerer det objekt der i banen er nærmest `Point`. Hvis der ingen Vertex, Edge eller Triangle, er returneres banen selv, altså et Map objekt. Derfor konstrueres et kald til select, der bør resultere i en af hver af disse situationer

`getProperties`

Ved først at sætte en mængde egenskaber på et Map trækkes de ud igen og sammenlignes med de oprindelige værdier.

Resultat

Alle dele i afprøvningen består.

13.3.2 Andre klassers afprøvning

Det samme er gjort for yderligere to klasser, Vertex og Exporter. Afprøvningen af disse kan findes i bilag E.2. Resultatet var at begge bestod. Andre klasser kunne afprøves efter samme mønster hvis der havde været tid.

13.4 Opsummering

Udfra den udførte afprøvning ses det, at langt det meste virker som det er tiltænkt. Unit test består alle, hvilket hovedsageligt skyldes, at de alle har været en del af selve udviklingen af klassen, de afprøver. Brugerafprøvningen påviser visse mangler i brugergrænsefladen.

Der mangler dog stadig den eksterne afprøvning. En hurtig gennemkørsel af programmet med JTest afslører manglende håndtering af RuntimeExceptions flere steder i programmet. Dette er et område til nærmere undersøgelse hvis tiden tillod det.

Del IV

Vurdering af produktet

Afsnit 14

Diskussion

I det følgende vil design og implementering af editoren og spillet blive diskuteret. Desuden vil de valg der er blevet truffet i projektet blive diskuteret.

14.1 Spillet

Det implementerede spil fungerede ikke som en helhed ved projektets afslutning. Klienten fungerede dog nok til at grundlæggende funktioner kunne testes.

14.1.1 Design

Spillet er designet over et MVC-mønster. Dette har gjort at der er selve spillogikken er separeret fra f.eks. fremvisning, brugerinteraktion osv. MVC-mønsteret har vist sig at være meget velegnet til spil, netop p.g.a. seperationen imellem de forskellige dele. Det har bl.a. lettet designet af serveren og klienten. På serveren kan man helt undlade at optegne spillet, uden at det påvirker spillets gang. På klienten kan man uden videre opdatere spillets model, uden at det giver nogle problemer.

Implementationen af View i spillet har også virket meget godt. Det er relativt let at indsætte et nyt view ind i spillet, uden at skulle ændre ved de eksisterende views.

Brugen af en fælles super-klasse for server/klient, til at håndtere hovedparten af netværks-kommunikationen har lettet implementationen af server og klient-kode. Det har ikke krævet mere end nogle få linjes kode at implementere klienten, når den først benyttede sig af den fælles netværkskode.

14.1.2 Implementeringen

I dette afsnit vil det blive vurderet hvorvidt det er lykkedes at implementere spillet tilfredsstillende. Der bliver taget udgangspunkt i kravspecifikationen for klient og server.

I server delen af spillet er stort set alle kravene blevet opfyldt. Der er dog visse steder implementationen halter. Det primære problen i server implementationen

lige er at beskedsystemet ikke fungere. Server kan godt oprette forbindelse via tcp og sende/modtage via udp. Men der er ingen måde at håndtere de besked serveren får vi udp. Det har betydet at det meste af spil-logikken enten ikke er implementeret, eller ikke har været testet. Al den grundlæggende funktionalitet, så som kollision-detektion, indlæsning af baner, og simpel fysik fungerer. Men da man ikke kan spille et egentligt spil er ting som point-givning hverken testet eller implementeret.

Klienten delen lider under de sammen problemmer som serveren. Den er således heller ikke i stand til at behandle de beskeder den får fra serveren, hvilket i praksis betyder, at den ikke kan modtage informationer om de andre spillere. Det er derfor ikke muligt at spille spillet sammen med andre spillere i øjeblikket. Endvidere er de sorte huller, der er nævnt i gameplay afsnittet ikke blevet implementeret. Disse er blevet undladt p.g.a tidsmangel og ikke fordi det ville være noget større problem at implementere det. Men hen mod slutningen af projektet var det nødvendigt at prioritere nogle ting ned. Da sorte huller ikke var essentiel for hverken klienten eller serverens funktionsevne, men blot var en del, der ville have fremmet spillets gameplay blev det nedprioriteret.

Generelt virker alt på nær netværket i den nuværende implementation af spillet. Dette skyldes hovedsageligt at beskeder ikke virker, men der har også vist sig at være småfejl i selve netværks-koden. Det skal dog bemærkes at der tilsyneladende kun er tale om fejl i programmet, og ikke mangler.

Hvis man overordnet set kal fremhæve noget, der kan betegnes som godt i programmet, kan bl.a. selv designet nævnes. Designet har gjort det det nemt så genbruge kode fra klient på serveren og omvendt. Envidere har de anvendte mønstre, hjulpet med til at spillet er lidt at udvide f.eks. med nye views. Det kan også fremhæves at selve den visuelle del af spillet er blevet vellykket da grafikken på skærmen stemmer overens med det forventede resultat.

Der er dog også vise elementer i spillet, der ikke fungere så godt som man kunne have håbet. Det gælder bl.a. kollision. Det er i øjeblikket ikke til at afgøre hvilken kant (`Edge`) man støder sammen med. Dette bevirker at man ikke kan lave realistiske kollisioner. Derudover er der er ikke nogen grafisk brugergrænseflade til spillet, hvilket gør det mindre bruger venligt. Det bevirker også at der er en del ting som f.eks. spillernavn og server adresse, der er fastkodet ind i programmet

Selvom der er fejl i spillet koden vuderer gruppen dog at det overordnet set er lykkedes at implementere spillet. Største delen af de krav, der er stillet til spillet er blevet opfyldt. Det er da også gruppens vurdering at den fejlbehæftede del samt yderligere brugerorienterede forbedringer vil kunne laves på ca 4 arbejdsdage.

14.2 Editor

Til spillet blev der udviklet en editor som et hjælpeværktøj til at designe baner. Editoren fungerede tilfredsstillende ved projektets afslutning.

14.2.1 Design

Editoren er lige som serveren udviklet efter MVC-mønsteret. Dette gør editoren let at udvide, hvis det en dag skulle være nødvendigt. Valget at bruge et Strategy-

mønster til at håndtere de forskellige tilstande i editoren, har vist sig at være yders fornuftig. Det er enddog meget let at tilføje nye tilstande til editoren. Dog kunne der med fordel være brugt f.eks. Command mønstret [Gam94] til menu funktioner.

14.2.2 Implementeringen

Vi har valgt at implementere editoren i Java, bl.a. fordi alle i gruppen på forhånd har et godt kendskab til Java. Dette har vist sig fornuftigt, da det har betydet at editoren på trods af dens mange egenskaber har været meget lettere at implementere end hvis den f.eks. var programmeret i C++. Sprog-valget har samtidigt givet platformsuafhængighed af editoren.

Hvis man ser på editoren med kravspecifikationen for øje er det umiddelbart lykkedes at implementere editoren fuldt tilfredsstillende. Den lever op til alle de krav der er stillet til den. Den overholder desuden MVC mønstret ret godt, hvilket gør den med at udvide.

En andet ting der skal fremhæves er de forskellige tilstande editoren kan befinde sig i. Lige nu kan editoren befinde sig i to tilstande. En hvor man kan tegne polygoner og en hvor man kan flyttte polygoner. For at gøre det muligt at udvide editoren med flere tilstande, er stategy [Gam94] mønstret blevet benyttet. Dette bevirker at man uden større problemer kan indsætte tilstande. Det kunne eksempelvis være ønskværdigt at farvelægge en polygon eller lægger en tekstur på en polygon. Et af kritikpunkterne i editoren er Brugerfladen. Den lever ikke helt op til de krav som man kunne forvente, f.eks. får man ikke noget visuelt feedback når man vælger en kant. Vi ser dog ikke dette som en større mangel, da editoren aldrig har være ment som andet end et hjælpemiddel til at skrive bane-filer, hvilket man må sige den fuldt lever op til.

14.3 Diskussion af valg

Det blev fra starten valgt at lave en editor til at opbyyge baner med. Selve ideen med at lave en bane editor er generelt en god ide til udvikling af spil. Udover at det er nemt for udvikleren af spillet at lave baner at teste spillet med, så kan evtuelle senere brugere og så drage nytte af en editor til at lave egne baner. Det kan dog argumenteres for at udvikling af en editor i forbindelse projektet har været skudt over målet. Det er kommet til at tage en for stor del af den tid, der var til tådighed i projektet. Det havde været mere hesigtsmæssigt, hvis man i stedet havde håndkodet nogle baner i stedet.

Endvidere kan det diskuteres, hvor hensigtsmæssigt der var at kode spillet i C++. Rent fagligt har det givet gruppen en indsigt i et nyt sporg som anvendes til mange store applikationer og spil. Herudover har det givet et indblik i en lidt anden tilgang til objekt orienteret programmering. Omvendt har det har det taget tid at sætte sig ind i et nyt sprog.

Hvis spillet havde været lavet i Java kunne hele problematikken med at finde biblioteker til understøttelse af platformsuafhængished have været undgået. Men det har trods alt været en meget lærerigt at anvende et andet programmeringssporg, tiltrods for de ulemper det har medført.

Afsnit 15

Konklusion

Vi må erkende at projektet har været for stort til at det realistisk kunne nås. Hvis et spil som Blaster, samt en editor skulle implementeres på kun 4 uger, ville det simpelthen kræve, at alt forløber uden problemer, da der ikke er afsat tid til at skulle løse uforudsete problemer.

Trods problemerne, virker store dele af spillet som beskrevet i kravspecifikationen og hele editoren virker som der er lagt op til i kravspecifikationen. Projektets varighed taget i betragtning mener vi at vi har nået meget.

Valget af C++ som sprog har betydet at vi ikke blev færdige, men det faglige udbytte har været meget højt, hvilket gør at vi mener projektet i mange henseender har været en succes.

Afsnit 16

Perspektivering

Første skridt i et evt. videre arbejde med projektet er at lave spillet færdig. Dvs. implementere den manglende netværkskode samt kollisionshåndtering i spillet. Dette vil gøre det muligt at spille spillet over netværk. Adressen til serveren er dog stadig angivet direkte i koden. En løsning kunne være en grafisk brugergrænseflade, der giver spilleren mulighed for at angive en adresse direkte fra spillet. I en sådan brugergrænseflade kunne man også indtaste spillers navn, farve etc.

Editoren kunne gøres mere brugervenlig. F.eks. med bedre visuel respons på brugerens manipulationer, samt flere muligheder for at manipulere banen. Muligheden for at vælge flere objekter af gangen kunne også tilføjes.

Når alt dette er på plads kan der f.eks. implementeres teksturer på polygoner, samt andre avancerede grafiske lækkerier.

Afsnit 17

Referencer

[Aho01] Lauri Ahonen. *TurboRaketti*. 2001.

http://www.kolumbus.fi/lauri.ahonen/tr.html

[Atk02] Jonathan Atkins. *SDL_net*. 2002.

http://jcatki.no-ip.org/SDL_net/SDL_net.pdf

[Cul99] Timothy R. Culp. *Industrial Strength Pluggable Factories*. C++ Report, october 1999 edition, 1999.

http://www.adtmag.com/joop/crarticle.asp?ID=1520

[Gam94] Richard Helm & Ralph Johnson & John Vlissides & Erich Gamma. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[Mey02] Bertrand Meyer. *Building bug-free O-O software: An introduction to Design by ContractTM*. Eiffel, 2002.

http://archive.eiffel.com/doc/manuals/technology/contract/

[Rob03] Pavel Vorobiev & Matthew Robinson. *Swing*. Manning Publications, 2nd edition edition, 2003.

http://manning.com/sbe/

[Ros01] James F. Kurose & Keith W. Ross. *Computer Networking A Top-Down Approach Featuring the Internet*. Addison-Wesley, 1 edition, 2001.

[vDSFJH] James Foley & Andries van Dam & Steven Feiner & John Hughes. *Computer Graphics, Principles and Practices*. Addison Wesley.

Afsnit 18

Supplerende Litteratur

[Ake02] Mark Segal & Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 1.4)*. Silicon Graphics, Inc., 2002.

[Eck01] Bruce Eckel. *Thinking in C++, Volume 1*. MindView, Inc., 2nd edition edition, 2001.

http://setlonnert.com/java/bruceeckel/

[Eck02] Bruce Eckel. *Thinking in C++, Volume 2*. MindView, Inc., 2nd edition edition, 2002.

http://setlonnert.com/java/bruceeckel/

[Ele02] The Mind Electric. *Electric XML 4.1 User Guide*. 2002.

http://www.themindelectric.com/docs/xml/guide/index.html

[Jia00] Xiaping Jia. *Object-oriented software development in Java: principles, patterns and frameworks*. Addison Wesley Longman, Inc, 2000.

[McC02] Mason McCuskey. *Why Pluggable Factories Rock My Multiplayer World*. GameDev.net, 2002.

http://www.gamedev.net/reference/articles/article841.asp

[Slo] Matt Slot. *Networking: Broadcast, Client/Server, Token Ring*. CodeWhore.com.

[Woo97] Jackie Neider & Tom Davis & Mason Woo. *OpenGL Programming Guide, The Official Guide to Learning OpenGL, Version 1.1*. Addison-Wesley Developers Press, second edition edition, 1997.

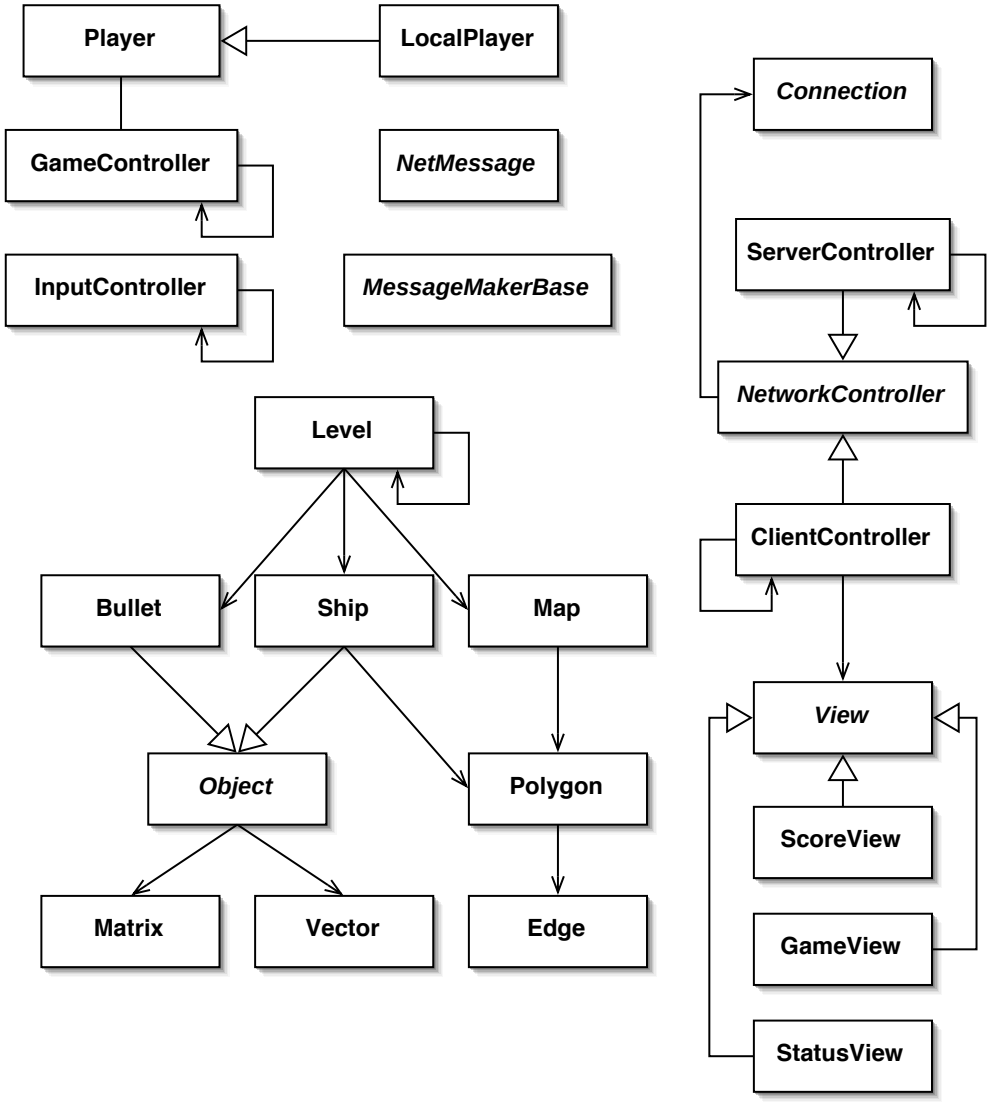
http://www.gamedev.net/download/redbook.pdf

Del V

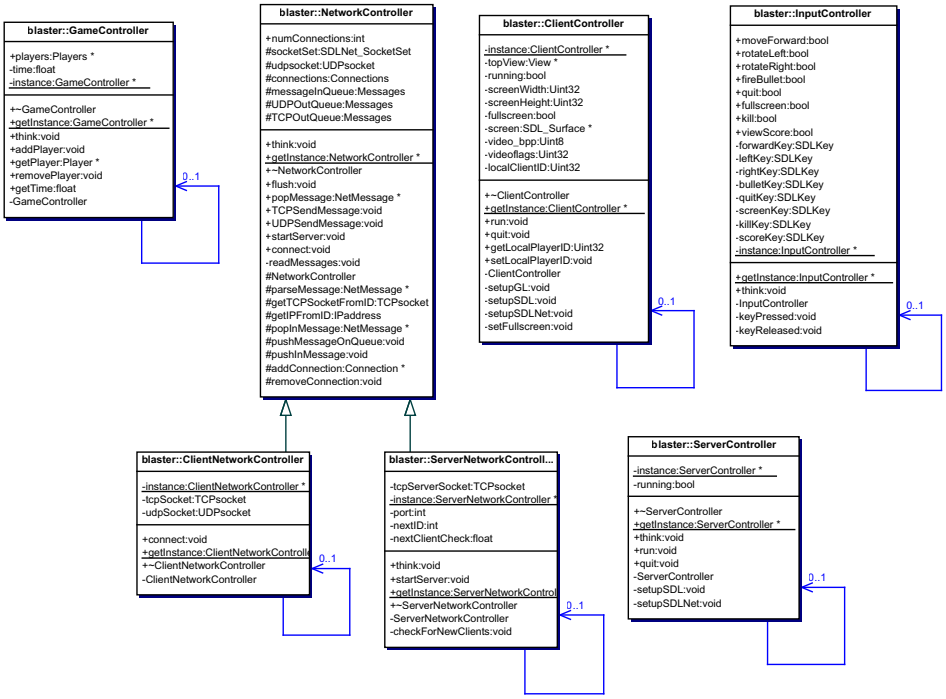
Appendiks

Appendiks A

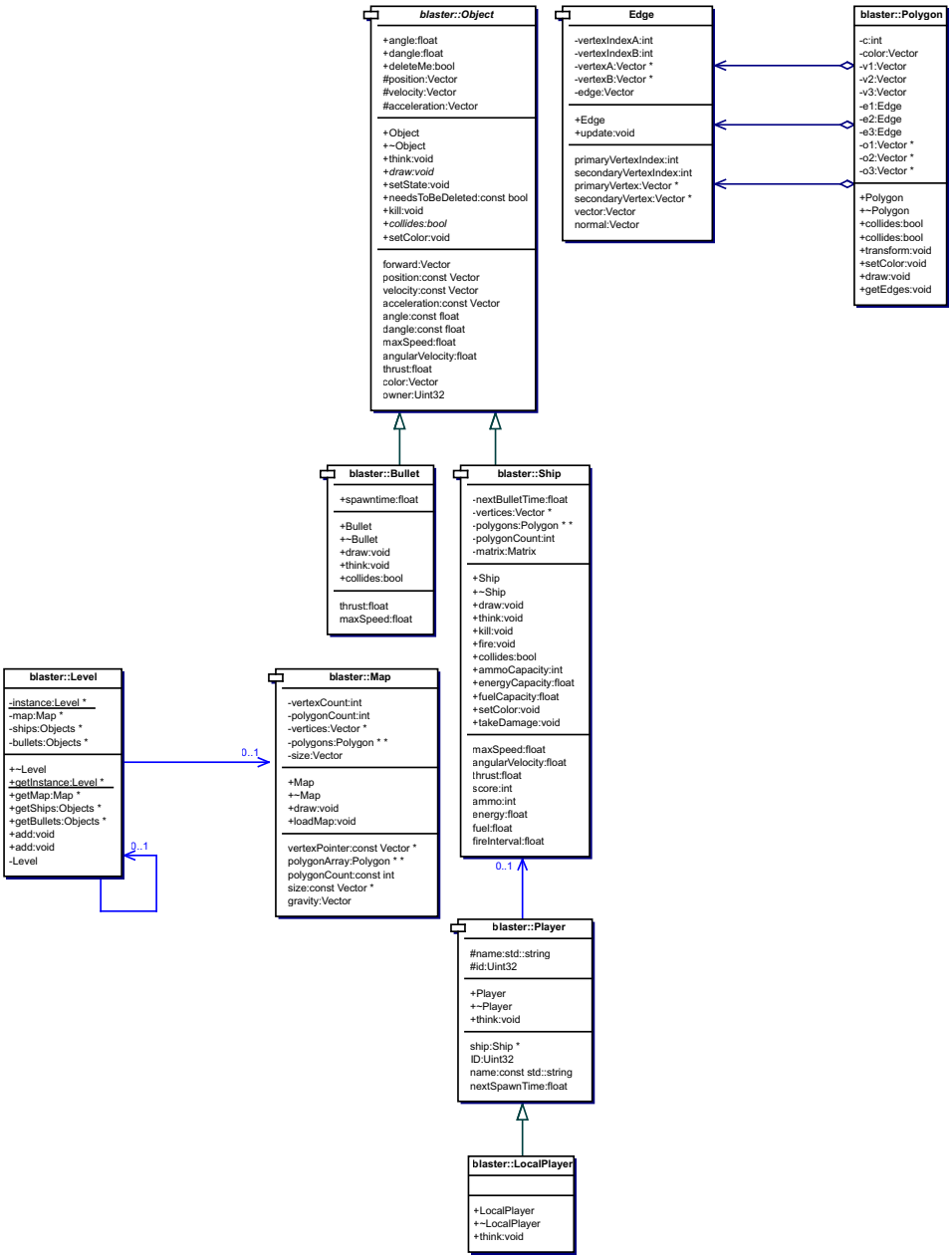
UML Diagrammer



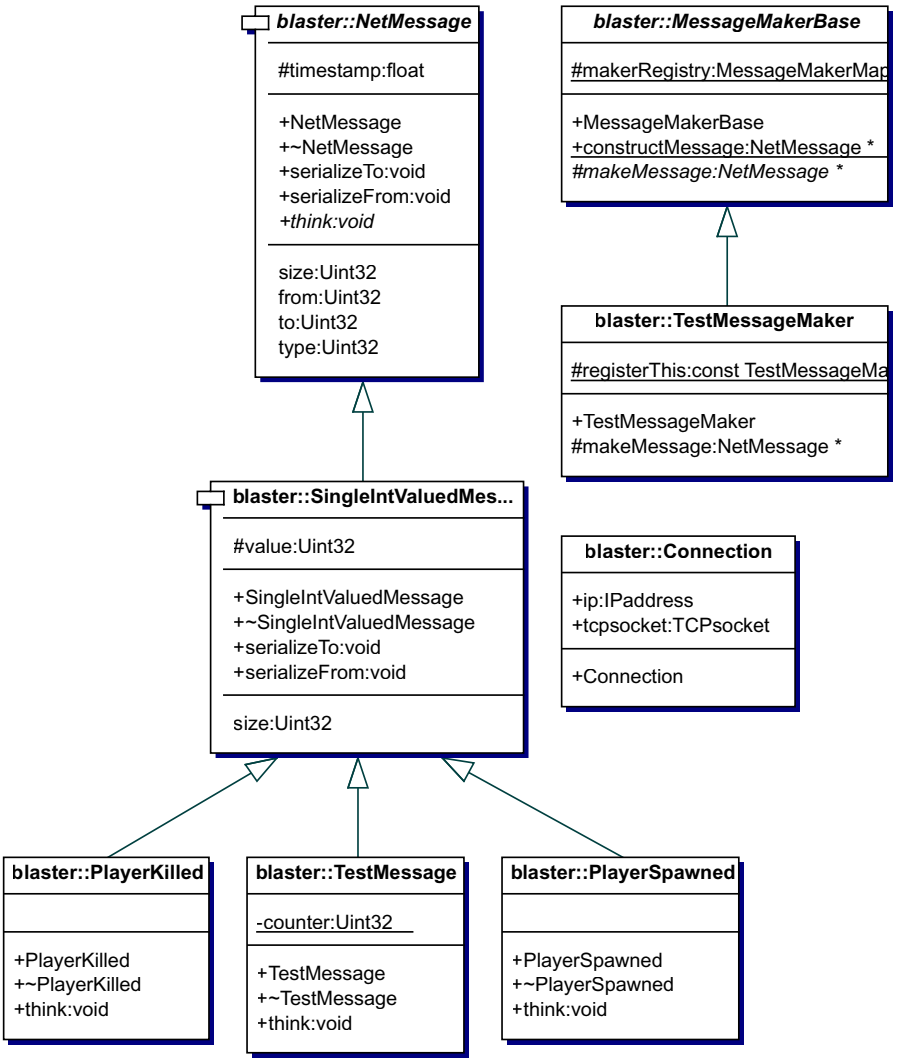
Figur A.1: Samlet oversigt over klient og server



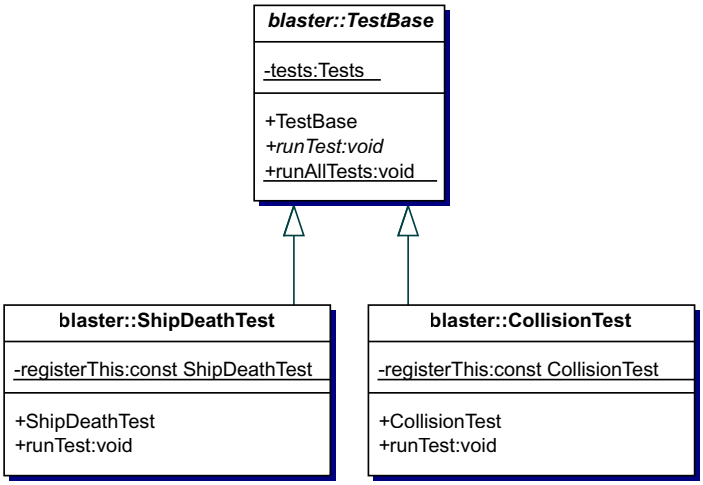
Figur A.2: UML diagram over controller delen



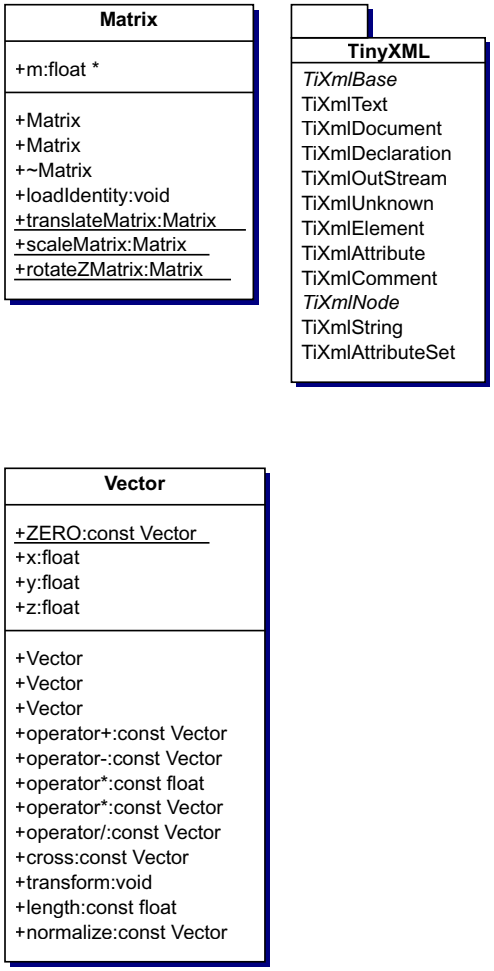
Figur A.3: UML diagram over model delen



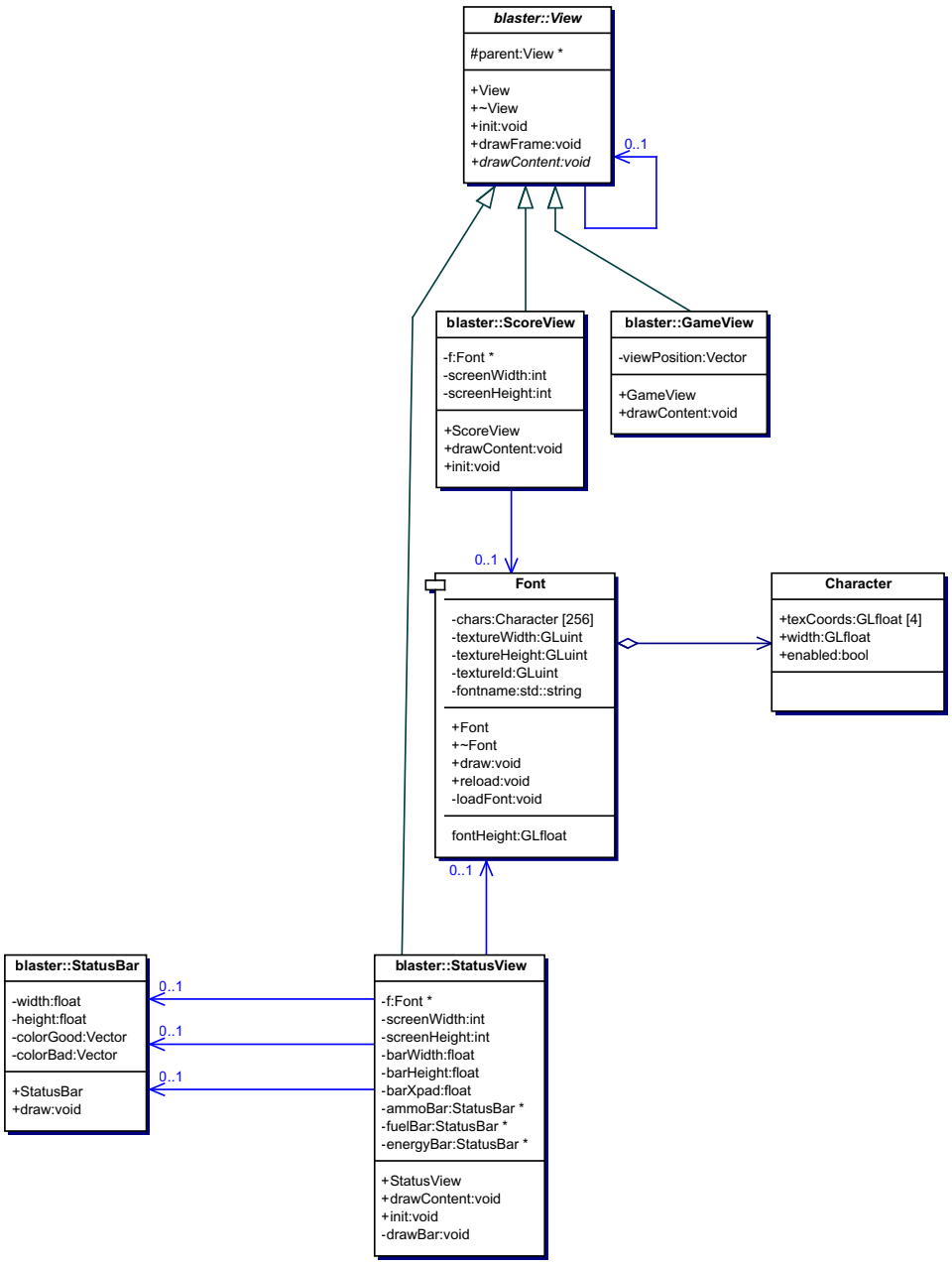
Figur A.4: UML diagram over netværks model delen



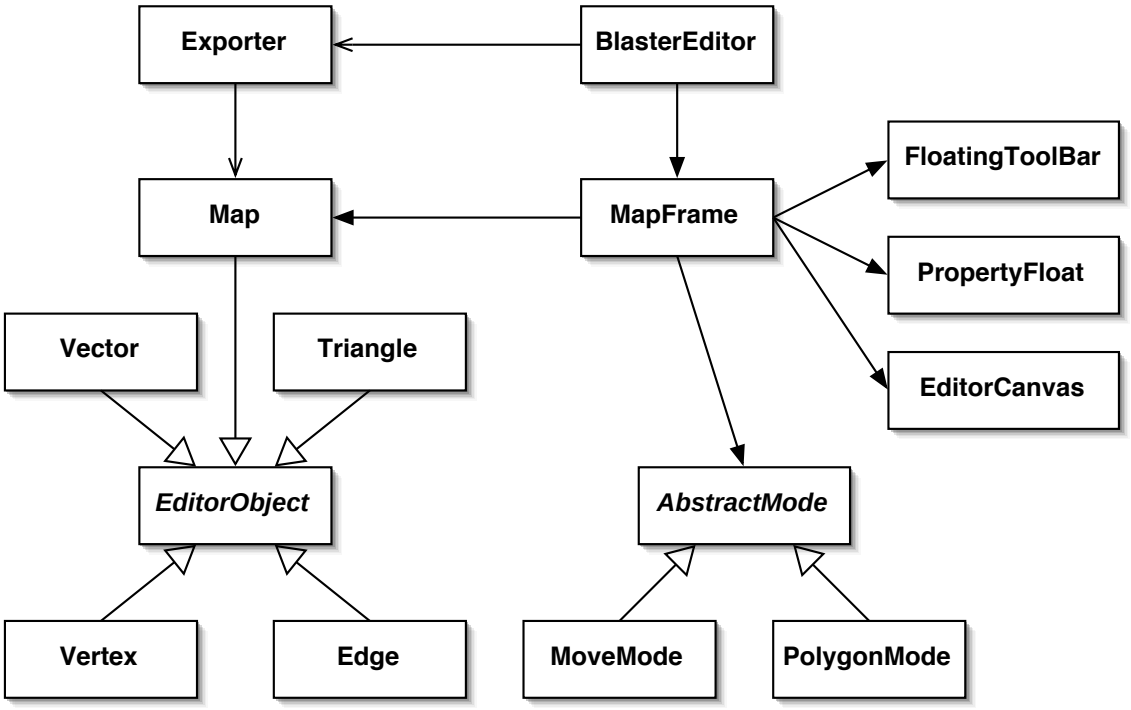
Figur A.5: UML diagram over test delen



Figur A.6: UML diagram over util delen



Figur A.7: UML diagram over views



Figur A.8: Klasse diagram over editoren

Appendiks B

Afprøvningsresultater

B.1 Test af spillet

B.1.1 Visuel afprøvning af spillet

Her følger skemaer med resultater af den visuelle test af spillet.

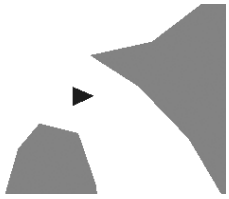
Start af spillet

Figur B.1: Spil ved start

Nr.	**Navn**	
1	Start af spilet	
Beskrivelse		
Spillet skal kunne starte		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Start spillet	Spillet starter og viser en bane, et skib, samt en række status-bjælker i bunden af skærmen.	Spillet starter og der ses en bane med et trekantet skib i midten. I bunden findes 3 status-bjælker, og en point-angivning.
Afprøvning bestået		Ja

Bevæg skibet frem

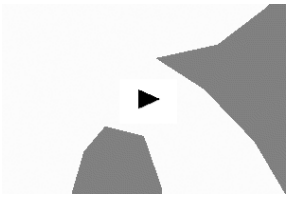
Figur B.2: Lige før bevægelse



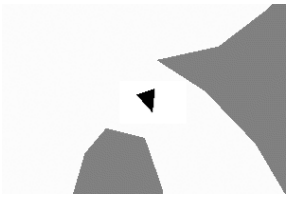
Figur B.3: Lige efter bevægelse

Nr.	**Navn**	
2	Bevæg skibet frem	
Beskrivelse		
Når der trykkes på fremdrifts-knappen (pil op), skal skibet flyve fremad, såfremt der er mere brændstof		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Se test 1	-	-
Tryk på fremdrifts-knappen	Skibet accelereres	Skibet accelererer, brændstofmåleren i bunden af skærmen falder.
Afprøvning bestået		Ja

Rotering skibet



Figur B.4: Lige før rotation



Figur B.5: Lige efter rotation

Nr.	**Navn**	
3	Rotering skibet	
Beskrivelse		
Når der trykkes på dreje-knapperne, (pil til højre og venstre) skal skal skibet dreje		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Se test 1	-	-
Tryk på højre piletast	Skibet drejer med uret	Skibet drejer med uret, om sin egen akse.
Tryk på venstre piletast	Skibet drejer mod uret	Skibet drejer mod uret, om sin egen akse.
Afprøvning bestået		Ja

Figur B.6: Lige før kollision

Figur B.7: Lige efter kollision

Kollision med bane

Nr.	**Navn**	
4	Kollision med bane	
Beskrivelse		
Ved et sammenstød imellem skibet og banen, skal skibet blive kastet tilbage, og miste energi.		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Se test 1	-	-
Bevæg skibet imod en kant i banen	Skibet bevæger sig imod banen	Skibet bevæger sig imod banen
Kollider med banen	Skibet bliver kastet tilbage, og mister energi	Skibets retning bliver vendt, og skibet mister omkring 150 i energi
Afprøvning bestået		Ja

Affyring af skud

Figur B.8: Lige før skud

Figur B.9: Lige efter skud

Nr.	Navn	
5	Affyring af skud	
Beskrivelse		
Når brugeren trykker på skyde-knappen, affyre skibet et skud, såfremt der er mere ammunition. Skuddet "lever" i 1 sekund hvorefter det forsvinder.		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Se test 1	-	-
Tryk på affyringsknappen (mellemrum)	Der affyres et skud der lever i 1 sekund	Der affyres et skud, da skuddet bevæger sig ud af skærmen på under 1 sekund kan det ikke afgøres hvor lang tid skuddet lever.
Afprøvning bestået		Delvist

Udtømning af brændstof

Nr.	Navn	
6	Udtømning af brændstof	
Beskrivelse		
Når skibet løber tør for brændstof, skal det ikke længere være istand til at bevæge sig accelerere.		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Se test 1	-	-
Skibet drives frem til det løber tør for brændstof	Skibet løber tør for brændstof	Skibet løber tør for brændstof.
Forsøg at ændre retning	Skibet er ude af stand til at ændre retning	Skibet drejer, men kan ikke ændre retning.
Afprøvning bestået		Ja

B.1.2 Eksterne tests

Disse tests undersøger om dele af implementationen overholder de eksternt definerede krav.

- Vedligeholdelse af spillerinformation
- Netværkskommunikation
- Kollisionshåndtering
- Ødelæggelse af spillerens skib

Vedligeholdelse af spillerinformation

Denne afprøvning undersøger om spillogikken er i stand til at vedligeholde information omkring spillere. Dette betyder i praksis, at der undersøges om der kan indsættes en spiller i spillet og denne senere findes ud fra samme id. Til sidst undersøges også om spilleren kan fjernes fra spillet.

Afprøvningen lykkedes.

Listing B.1: Pseudo-kode for vedligeholdelse af spillerinformation

```
opret spiller med givet id

tilføj spiller til spillet

find spilleren i spillet med det givne id

forudsæt at det er den samme spiller som blev
   indsat

fjern spiller med det givne id fra spillet

forudsæt at spilleren ikke længre er med i
   spillet
```

Netværkskommunikation

Denne afprøvning afprøver om det er muligt at oprette en forbindelse mellem en klient og en server. Derefter sendes en besked. Hvis beskeden bliver modtaget korrekt, bliver der sat et flag som testen aflæser.

Det forventede resultat er, at klienten er i stand til at forbinde til serveren, og at flaget vil blive sat når serveren modtager beskeden.

Afprøvningen lykkedes ikke. Det lykkedes at forbinde klienten til serveren, men beskeden blev ikke modtaget. Dette skyldes, at MessageMaker og TestMessage-Maker ikke er færdigimplemeteret.

Listing B.2: Pseudo-kode for test af kollision

```
opret server på en given port

opret klient

forbind klienten til serveren på den givne
   port

forudsæt at der er oprettet en forbindelse

opret en testmessage

send beskeden fra klienten til serveren

lad serveren modtage de beskeder der måtte
   være

forudsæt at testflaget er sat til sand
```

Kollisionshåndtering

Denne afprøvning undersøger om kollisionsalgoritmen er i stand til at konstatere om to polygoner overlapper. Afprøvningen er implementeret i CollisionTest `klassen`.

Der undersøges om to polygoner overlapper når de ligger oven i hinanden. Derefter undersøges om de overlapper hvis de har to punkter der overlapper, men ellers ikke overlapper med hinanden. Til sidst roteres det ene polygon således at de ikke burde overlappe. Det forventede resultat er, at polygonerne konstateres at overlappe når de ligger oven i hinanden samt når de deler et punkt. Når det ene polygon roteres således at de ikke længre deler et punkt, forventes det at de ikke overlapper.

Afprøvningen viste at algoritmen var i stand til at konstatere når to polygoner overlapper.

Listing B.3: Pseudo-kode for test af kollision

```
opret to polygoner

placer polygonerne oven i hinanden

forudsæt at polygonerne overlapper

flyt den ene polygon således at et af
   punkterne ligger oven i et punkt på den
   anden polygon, men resten ikke overlapper

forudsæt at polygonerne overlapper

roter det ene polygon således at polygonerne
   ligger spejlvendt i forhold til hinanden

forudsæt at polygonerne ikke overlapper
```

subsubsectionØdelæggelse af spillerens skib

Denne afprøvning undersøger om spillerens skib bliver genskabt tre sekunder efter at det ødelægges. Afprøvningen er implementeret i `ShipDeathTest` klassen, som er illustreret i liste B.4.

En spiller tilføjes til spillet, hvorefter spillerens skib ødelægges. Det forventede resultat er, at spilleren har fået et nyt skib tre sekunder efter det gamle blev ødelagt.

Afprøvningen var en success.

Listing B.4: Pseudo-kode for ødelæggelse af spillerens skib

```
opret spiller med given id

tilføj spiller til spillet

lad spillogikken give spilleren et skib

forudsæt at spilleren har et skib

tilføj spillerens skib nok skade til at det
   ødelægges

lad spillogikken fjerne spillerens skib

forudsæt at spilleren ikke har noget skib

lad tiden skride mindre end tre sekunder frem

forudsæt at spilleren stadig ikke har noget
   skib

lad tiden skride frem til over tre sekunder
   efter skibets ødelæggelse

lad spillogikken give spilleren et skib

forudsæt af spilleren har et skib
```

B.2 Test af editoren

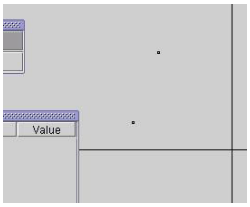
B.2.1 Visuel afprøvning af editor

Her følger skemaer med resultater af den visuelle test af editoren.

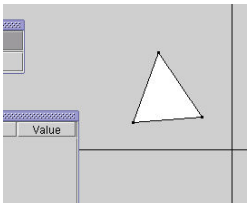
Tre punkter danner polygon

Nr.	**Navn**
1	Tre punkter danner polygon
Platform	MacOS X
Beskrivelse	
Ved at afsætte tre punkter efter hinanden i editoren dannes en udfyldt trekant umiddelbart efter det tredje punkt er afsat.	

Trin beskrivelse	**Forventet resultat**	**Resultat**
Åbn editoren	Editoren startet, klar til input	Editoren er klar til input
Lav en ny bane	Et nyt bane vindue i editoren	Bane vindue klar til input
Afsæt et punkt i banen	Banen indeholder et punkt og der hvor der blev trykket ses nu en lille firkant	Et lille punkt ses i bane vinduet
Afsæt endnu et punkt lidt nord vest for det første punkt	Endnu et punkt i editoren	En lille firkant til
Afsæt endnu et punkt øst for det andet punkt	En hvid trekant dannes med sorte kanter og endnu en lille firkant	Der dannes en trekant
Afprøvning bestået		Ja, se evt figur B.10, B.11



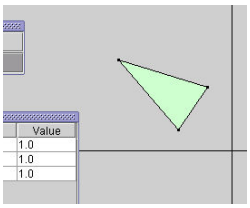
Figur B.10: To punkter afsat



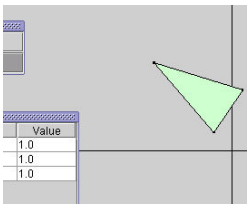
Figur B.11: Trekant dannet

Vælg og flyt polygon

Nr.	**Navn**	
Platform	Windows 2000	
2	Valg og flyt af polygon	
Beskrivelse		
Ved at vælge en polygon og trække i den kan man flytte den rundt i banen		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Følg scenarie 1	-	-
Stil markøren over polygonen og tryk på polygonen	Polygonen vælges og egenskabs paletten viser egenskaberne for polygonen (farve komponenterne rød, grøn og blå)	Polygonet vælges og skifter farve og egenskaberne vises i egenskabspaletten
Træk musen mens knappen holdes nede	Polygonen følger med musen	Polygonen flyttes med musen rundt
Afprøvning bestået		Ja, se evt figur B.12, B.13



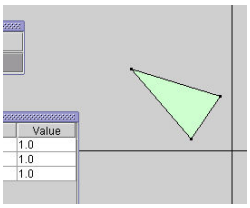
Figur B.12: Polygon valgt



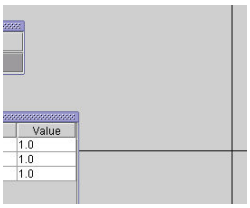
Figur B.13: Polygon flyttet

Slet polygon

Nr.	**Navn**	
Platform	Windows 2000	
3	Slet polygon	
Beskrivelse		
Vælges en polygon og der trykkes delete slettes polygonen		
Trin beskrivelse	**Forventet resultat**	**Resultat**
Følg scenarie 1 Vælg polygonet Tryk delete	- Polygon valgt Polygonet slettes	- Polygon valgt Polygonen forsvinder fra banen
Afprøvning bestået		Ja, se evt figur B.14, B.15



Figur B.14: Polygon valgt

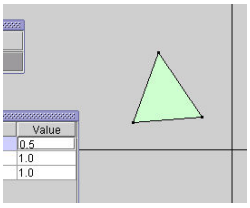


Figur B.15: Polygon slettet

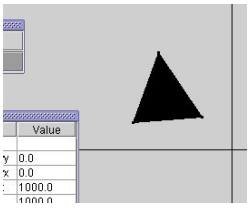
Skift farve

Nr.	**Navn**
4	Skift farve
Platform	MacOS X
Beskrivelse	
Ved at ændre i farve egenskaberne for en polygon skifter polygonen farve	

Trin beskrivelse	**Forventet resultat**	**Resultat**
Følg scenarie 1	-	-
Vælg polygonen og skift farve	Polygonen skifter farve	Polygonen bliver rødt, men først efter at polygonen ikke længere er valgt. Måske burde farven der angiver at en polygon er valgt være en toning af polygonens farve
Afprøvning bestået		Ja, men måske bør der ses på hvordan en polygon vises som værende valgt, se evt figur B.16, B.17



Figur B.16: Polygon valgt

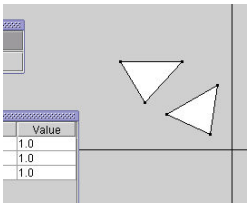


Figur B.17: Polygonen farvet

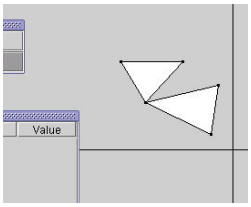
Sammensmeltning af to punkter

Platform	MacOS X
Nr.	**Navn**
5	Sammensmeltning af to punkter
Beskrivelse	
Ved at trække et punkt hen oven i et andet punkt skal de to samles til et punkt	

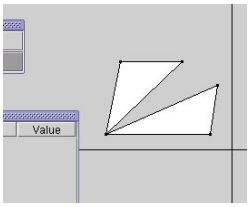
Trin beskrivelse	**Forventet resultat**	**Resultat**
Følg scenarie 1	-	-
Gentag scenarie 1 og lav endnu et polygon	-	-
Vælg det punkt i det ene polygon der er nærmest den anden polygon	Egenskabspaletten opdateres og punktet er valgt	Punktet er valgt (tror jeg nok)
Træk det valgte punkt over til den anden polygons nærmeste punkt	Punktet flyttes og polygonerne omformes	Polygonen ændrer form
Vælg nu punktet igen	Punktet valgt	Punktet er valgt
Træk i punktet	Begge polygoner skifter form da punktet er smeltet sammen	Begge polygoner skifter form
Afprøvning bestået		Ja, se evt figur B.18, B.19, B.20



Figur B.18: To polygoner lavet



Figur B.19: Punkter sat sammen

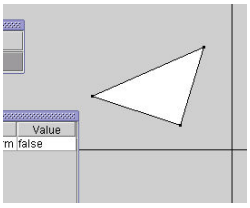


Figur B.20: Punkter rykket sammen

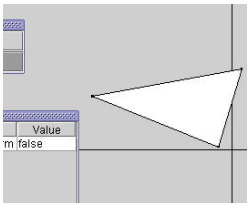
Flytning af kant

Nr.	**Navn**
6	Flytning af kant
Platform	MacOS X
Beskrivelse	
Ved at vælge en kant og trække i den kan man flytte de to punkter som kanten består af	

Trin beskrivelse	**Forventet resultat**	**Resultat**
Følg scenarie 1	-	-
Tryk på en kant	Kanten vælges og egenskabs paletten opdateres	Kanten ser ud til at være valgt da egenskabspaletten opdateres
Træk musen mens knappen holdes nede	Kanten flyttes	Kanten flyttes og polygonen opdateres løbende
Afprøvning bestået		Ja, men der burde være visuel feedback når den vælges, se evt figur B.21, B.22



Figur B.21: Kant valgt



Figur B.22: Kant flyttet

B.2.2 Unit test af editor

Unit Tests Results

Designed for use with JUnit and Ant.

Summary

Tests	Failures	Errors	Success Rate	Time(s)
8	0	0	100.00%	5.042

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

Name	Tests	Errors	Failures	Time(s)
dk.ruc.blaster.export	1	0	0	2.257
dk.ruc.blaster.model	7	0	0	2.785

Package dk.ruc.blaster.export

Name	Tests	Errors	Failures	Time(s)
ExporterTest	1	0	0	2.257

Show/Hide Properties

Back to top

Package dk.ruc.blaster.model

Name	Tests	Errors	Failures	Time(s)
MapTest	5	0	0	1.629

Show/Hide Properties

VertexTest	2	0	0	1.156

Show/Hide Properties

Back to top

TestCase ExporterTest

Name	Status	Type	Time(s)
testExport	Success		1.621

Back to top

TestCase MapTest

Name	Status	Type	Time(s)
testSelectPoint	Success		0.894
testGetTrianglesInRect	Success		0.084
testGetVerticesInRect	Success		0.010
testGetProperties	Success		0.003
testBounds	Success		0.004

Back to top

TestCase VertexTest

Name	Status	Type	Time(s)
testGetNumberOfAttachedPolygons	Success		0.172
testMove	Success		0.012

Back to top

Appendiks C

Klient og server kilde kode

ClientController.cpp Page 1/4

```
#include "ClientController.h"
#include "GameController.h"
#include "ClientNetworkController.h"
#include "GameView.h"
#include "StatusView.h"
#include "ScoreView.h"
#include "Vector.h"
#include "InputController.h"
#include "TestMessage.h"
#include <iostream>

#include "LocalPlayer.h"

using namespace blaster;
using namespace std;

ClientController::ClientController( void )
{
    video_bpp = 0;
    videoflags = SDL_SWSURFACE | SDL_OPENGL;

    screenWidth = 640;
    screenHeight = 480;

    fullscreen = false;

    running = true;

    setupSDL(  );
    setupSDLNet(  );
    setupGL(  );

    View *gameView = new GameView( NULL );
    View *statusView = new StatusView( gameView );
    View *scoreView = new ScoreView( statusView );

    topView = scoreView;
    topView->init(  );

}

ClientController::~ClientController( void )
{
    delete topView;

    SDLNet_Quit(  );
    SDL_Quit(  );
}


ClientController *ClientController::instance = NULL;

ClientController *ClientController::getInstance( void )
{
    if ( instance == NULL )
    {
        instance = new ClientController(  );
    }

    return instance;
}

// turn this on if you want to run the client without a server
```

ClientController.cpp Page 2/4

```
#define NOSERVER

// turn this on to create another player
#define FAKEPLAYER

void ClientController::run( void )
{
    // Hard-code the server location for now
    Uint16 port = 1337;

    std::string address = "192.168.0.41";

    long launchtime = SDL_GetTicks(  );
    long oldtime = 0;
    long newtime = 0;

    float time = 0.0f;
    float dt = 0.0f;

    InputController *input;
#ifndef NOSERVER
    ClientNetworkController *network =
        ClientNetworkController::getInstance(  );

    network->connect( address, port );
#endif

    // Get player id from server at some point
    Player * player = new LocalPlayer(  );

    player->setID( 42 );

    setLocalPlayerID( 42 );
    GameController::getInstance(  )->addPlayer( player );

#ifdef FAKEPLAYER
    Player *p2 = new Player(  );

    p2->setID( 1337 );
    GameController::getInstance(  )->addPlayer( p2 );
#endif

    while ( running )
    {
        newtime = SDL_GetTicks(  ) - launchtime;
        dt = ( float ) ( newtime - oldtime ) * 0.001f;
        time = ( float ) newtime *0.001f;

        oldtime = newtime;

        input = InputController::getInstance(  );
        input->think(  );

#ifndef NOSERVER
        network = ClientNetworkController::getInstance(  );
        network->think( time, dt );

        network->UDPSendMessage( new TestMessage( time ) );
#endif

        // Move this elsewhere?
        if ( input->quit )
        {
```

```
            quit(  );
        }
        if ( input->fullscreen )
        {
            setFullscreen( !fullscreen );
            setupGL(  );
            topView->init(  );
        }

        // Update the game world
        GameController::getInstance(  )->think( time, dt );

        // Draw the screen
        topView->drawFrame(  );

#ifndef NOSERVER
        // Send any messages that we've queued during the frame
        network->flush(  );
#endif
    }

    SDL_ShowCursor( 1 );
}


void ClientController::setupSDL(  )
{
    video_bpp = 0;
    videoflags = SDL_SWSURFACE | SDL_OPENGL;

    if ( SDL_Init( SDL_INIT_VIDEO | SDL_INIT_TIMER ) < 0 )
    {
        fprintf( stderr, "Couldn't initialize SDL: %s\n", SDL_GetError(  ) );
        exit( 1 );
    }
    SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 8 );
    SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 8 );
    SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 8 );
    SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
    SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );

    SDL_ShowCursor( 0 );

    setFullscreen( fullscreen );
}

void ClientController::setFullscreen( bool value, bool alreadyFailed )
{
    screen =
        SDL_SetVideoMode( screenWidth, screenHeight, video_bpp,
                          videoflags | ( value ? SDL_FULLSCREEN : 0 ) );

    if ( screen == NULL )
    {
        if ( !alreadyFailed )
        {
            std::cerr << "setFullscreen(" << value << ") failed." << std::
                endl;
            setFullscreen( !value, true );
            return;
        }

        fprintf( stderr, "Unable to create %dx%dx%d screen: %s\n",
```



```
                 screenWidth, screenHeight, video_bpp, SDL_GetError(  ) );
        SDL_Quit(  );
        exit( 2 );
    }

    fullscreen = value;
}

void ClientController::setupGL( void )
{
    glClearColor( 0.0f, 0.0f, 0.0f, 0.0f );

    glShadeModel( GL_SMOOTH );

    glEnable( GL_BLEND );
    glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
    glEnableClientState( GL_VERTEX_ARRAY );

    glViewport( 0, 0, screenWidth, screenHeight );

    glMatrixMode( GL_PROJECTION );
    glLoadIdentity(  );

    float drawWidth = 500.0f;
    float drawHeight =
        ( ( float ) screenHeight / ( float ) screenWidth ) * drawWidth;

    std::
        cout << "drawWidth: " << drawWidth << ", drawHeight: " << drawHeight
        << std::endl;

    glOrtho( -drawWidth / 2, drawWidth / 2, drawHeight / 2, -drawHeight / 2,
             -1.0, 1.0 );

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity(  );
}

void ClientController::setupSDLNet( void )
{
    // initialize SDL_net
    if ( SDLNet_Init(  ) == -1 )
    {
        printf( "SDLNet_Init: %s\n", SDLNet_GetError(  ) );
        exit( 2 );
    }
}

int main( int argc, char **argv )
{
    cout << "her er en main" << endl;


    // Resolve the argument into an IPaddress type

    ClientController *client;

    client = ClientController::getInstance(  );
    client->run(  );

    return 0;
}
```

ClientController.h

Page 1/1

```
#ifndef CLIENTCONTROLLER_H
#define CLIENTCONTROLLER_H

#include "SDL.h"
#include "SDL_net.h"
#include "View.h"
#include <string>

namespace blaster
{
    class ClientController
    {
      public:
        ~ClientController( void );
        static ClientController *getInstance( void );
        void run( void );
        void quit( void )
        {
            running = false;
        }

        Uint32 getLocalPlayerID( void )
        {
            return localClientID;
        }
        void setLocalPlayerID( Uint32 i )
        {
            localClientID = i;
        }

      private:
        ClientController( void );
        static ClientController *instance;
        void setupGL( void );
        void setupSDL( void );
        void setupSDLNet( void );
        void setFullscreen( bool value, bool alreadyFailed = false );

        View *topView;

        bool running;
        Uint32 screenWidth, screenHeight;
        bool fullscreen;
        SDL_Surface *screen;
        Uint8 video_bpp;
        Uint32 videoflags;

        Uint32 localClientID;
    };

}

#endif
```

ClientNetworkController.cpp

Page 1/2

```
#include "ClientNetworkController.h"
#include <iostream>
#include "SDL_net.h"

using namespace blaster;
using namespace std;

bool ClientNetworkController::connect( const std::string & address,
                                       Uint16 port )
{
    TCPsocket tcpSocket;

    IPaddress ip;

    int addressLength = address.size(  );

    char adr[addressLength + 1];

    strcpy( adr, address.c_str(  ) );

    if ( SDLNet_ResolveHost( &ip, adr, port ) == -1 )
    {
        printf( "SDLNet_ResolveHost: %s\n", SDLNet_GetError(  ) );
        return false;
    }

    // open the tcp socket to the server
    tcpSocket = SDLNet_TCP_Open( &ip );
    if ( !tcpSocket )
    {
        printf( "SDLNet_TCP_Open: %s\n", SDLNet_GetError(  ) );
        return false;
    }

    // open the tcp server socket
    udpsocket = SDLNet_UDP_Open( 0 );

    if ( !udpsocket )
    {
        cerr << "SDLNet_UDP_Open: " << SDLNet_GetError(  ) << endl;
        return false;
    }

    // bind server address to channel 0
    if ( SDLNet_UDP_Bind( udpsocket, 0, &ip ) == -1 )
    {
        printf( "SDLNet_UDP_Bind: %s\n", SDLNet_GetError(  ) );
        return false;
    }

    addConnection( 0, tcpSocket, ip );

    return true;
}

void ClientNetworkController::disconnect( void )
{
    removeConnection( 0 );
}

ClientNetworkController *ClientNetworkController::instance = NULL;

ClientNetworkController *ClientNetworkController::getInstance( void )
```

ClientNetworkController.cpp

Page 2/2

```
{
    if ( instance == NULL )
    {
        instance = new ClientNetworkController(  );
    }

    return instance;
}
```

ClientNetworkController.h

Page 1/1

```
#ifndef CLIENTNETWORKCONTROLLER_H
#define CLIENTNETWORKCONTROLLER_H

#include <string>
#include "SDL.h"

#include "NetworkController.h"

namespace blaster
{

    class ClientNetworkController:public NetworkController
    {
      public:
        static ClientNetworkController *getInstance( void );

        virtual bool connect( const std::string & address, Uint16 port );
        virtual void disconnect( void );

        virtual ~ ClientNetworkController( void )
        {
        }

      private:
          ClientNetworkController( void ):NetworkController(  )
        {
        }
        static ClientNetworkController *instance;

        TCPsocket tcpSocket;
        UDPsocket udpSocket;
    };
}

#endif
```

GameController.cpp


```
#include "GameController.h"

#include "Level.h"
#include "Player.h"

#include <iostream>

using namespace blaster;
using namespace std;

GameController::GameController( void )
{
    players = new Players(  );
}

GameController::~GameController( void )
{
    delete players;
}

void GameController::think( const float time, const float dt )
{
    this->time = time;

    Level *level = Level::getInstance(  );

    Objects *bullets = level->getBullets(  );
    Objects *ships = level->getShips(  );

    for ( Players::iterator i = players->begin(  );
          i != players->end(  ) && !players->empty(  ); i++ )
    {
        ( *i ).second->think( time, dt );
    }

    for ( Objects::iterator i = ships->begin(  );
          i != ships->end(  ) && !ships->empty(  ); i++ )
    {
        Object *b = ( *i );

        if ( b->needsToBeDeleted(  ) )
        {
            i = ships->erase( i );

            delete b;

            continue;
        }

        b->think( time, dt );
    }

    int polyCount = level->getMap(  )->getPolygonCount(  );
    Polygon **polygons = level->getMap(  )->getPolygonArray(  );

    for ( int i = 0; i < polyCount; i++ )
    {
        Polygon *polygon = polygons[i];

        for ( Objects::iterator it = ships->begin(  ); it != ships->end(  );
              it++ )
        {
            Ship *s = ( Ship * ) * it;
```

GameController.cpp


```
            if ( s->collides( polygon ) )
            {
                // Let damage be determined by speed
                float speed = s->getVelocity(  ).length(  );

                if ( speed < 10.0f )
                {
                    s->setVelocity( Vector::ZERO );
                }
                else
                {
                    s->takeDamage( speed );

                    // Get the reflection vector from the edge that we hit
                    s->setVelocity( Vector::ZERO - s->getVelocity(  ) );
                }

                s->setColor( 1.0f, 0.0f, 0.0f );
            }
            else
            {
                s->setColor( 0.0f, 1.0f, 1.0f );
            }
        }
    }

    for ( Objects::iterator i = bullets->begin(  );
          i != bullets->end(  ) && !bullets->empty(  ); i++ )
    {
        Object *b = ( *i );

        if ( b->needsToBeDeleted(  ) )
        {
            i = bullets->erase( i );
            delete b;

            continue;
        }

        b->think( time, dt );
    }
    for ( int i = 0; i < polyCount; i++ )
    {
        Polygon *polygon = polygons[i];

        for ( Objects::iterator i = bullets->begin(  );
              i != bullets->end(  ) && !bullets->empty(  ); i++ )
        {
            Bullet *b = ( Bullet * ) * i;

            if ( b->collides( polygon ) )
            {
                b->kill(  );
            }
        }
    }

    for ( Objects::iterator i = ships->begin(  );
          i != ships->end(  ) && !ships->empty(  ); i++ )
    {
        for ( Objects::iterator j = bullets->begin(  );
              j != bullets->end(  ) && !bullets->empty(  ); j++ )
```

GameController.cpp


```
        {
            Ship *ship = ( Ship * ) * i;
            Object *bullet = *j;

            if ( ship->getOwner(  ) != bullet->getOwner(  ) )
            {
                Polygon **polys = ship->getPolygonArray(  );
                int polycount = ship->getPolygonCount(  );
                bool collided = false;

                for ( int k = 0; k < polycount; k++ )
                {
                    if ( bullet->collides( polys[k] ) )
                    {
                        collided = true;
                        break;
                    }
                }

                if ( collided )
                {
                    ship->takeDamage( 75 );
                    bullet->kill(  );
                }
            }
        }
    }
}

GameController *GameController::instance = NULL;

GameController *GameController::getInstance( void )
{
    if ( instance == NULL )
    {
        instance = new GameController(  );
    }

    return instance;
}

void GameController::addPlayer( Player * p )
{
    players->insert( std::make_pair( p->getID(  ), p ) );
}

Player *GameController::getPlayer( Uint32 id )
{
//  return (*players)[id];
    Players::iterator i = players->find( id );

    return ( *i ).second;
}

void GameController::removePlayer( Uint32 id )
{
    Players::iterator i = players->find( id );

    players->erase( i );

    delete( *i ).second;
}
```

GameController.h


```
#ifndef GAMECONTROLLER_H
#define GAMECONTROLLER_H

#include "Player.h"
#include "SDL.h"

#include <map>

namespace blaster
{
    typedef std::map < Uint32, Player * >Players;

    class GameController
    {
      public:
        ~GameController( void );

        static GameController *getInstance( void );
        void think( const float time, const float dt );

        Players *players;

        void addPlayer( Player * p );
        Player *getPlayer( Uint32 id );
        void removePlayer( Uint32 id );

        float getTime( void )
        {
            return time;
        }

      private:
          GameController( void );
        float time;

        static GameController *instance;
    };

}

#endif
```

InputController.cpp


```
#include "InputController.h"

using namespace blaster;

InputController *InputController::instance = NULL;

InputController::InputController( void )
{
    // Key bindings
    forwardKey = SDLK_UP;
    leftKey = SDLK_LEFT;
    rightKey = SDLK_RIGHT;
    bulletKey = SDLK_SPACE;
    quitKey = SDLK_ESCAPE;
    screenKey = SDLK_RETURN;
    killKey = SDLK_k;
    scoreKey = SDLK_TAB;

    // Key state
    moveForward = false;
    rotateLeft = false;
    rotateRight = false;
    fireBullet = false;
    quit = false;
    fullscreen = false;
    kill = false;
    viewScore = false;
}

InputController *InputController::getInstance( void )
{
    if ( instance == NULL )
    {
        instance = new InputController(  );
    }

    return instance;
}

void InputController::think( void )
{
    SDL_Event event;

    while ( SDL_PollEvent( &event ) )
    {
        switch ( event.type )
        {
        case SDL_KEYDOWN:
            {
                keyPressed( event.key.keysym.sym );
                break;
            }

        case SDL_KEYUP:
            {
                keyReleased( event.key.keysym.sym );
                break;
            }

        default:
            break;
        }
```

InputController.cpp


```
    }
}

void InputController::keyPressed( const SDLKey & key )
{
    // C++ doesn't support switches with non-constant values.
    if ( key == forwardKey )
    {
        moveForward = true;
    }
    else if ( key == leftKey )
    {
        rotateLeft = true;
    }
    else if ( key == rightKey )
    {
        rotateRight = true;
    }
    else if ( key == bulletKey )
    {
        fireBullet = true;
    }
    else if ( key == quitKey )
    {
        quit = true;
    }
    else if ( key == screenKey )
    {
        fullscreen = true;
    }
    else if ( key == killKey )
    {
        kill = true;
    }
    else if ( key == scoreKey )
    {
        viewScore = true;
    }
}

void InputController::keyReleased( const SDLKey & key )
{
    // C++ doesn't support switches with non-constant values.
    if ( key == forwardKey )
    {
        moveForward = false;
    }
    else if ( key == leftKey )
    {
        rotateLeft = false;
    }
    else if ( key == rightKey )
    {
        rotateRight = false;
    }
    else if ( key == bulletKey )
    {
        fireBullet = false;
    }
    else if ( key == quitKey )
    {
        quit = false;
    }
```

InputController.cpp

Page 3/3

```
    else if ( key == screenKey )
    {
        fullscreen = false;
    }
    else if ( key == killKey )
    {
        kill = false;
    }
    else if ( key == scoreKey )
    {
        viewScore = false;
    }

}
```

InputController.h

Page 1/1

```
#ifndef INPUTCONTROLLER_H
#define INPUTCONTROLLER_H

#include "SDL.h"

namespace blaster
{

    class InputController
    {
      public:
        static InputController *getInstance( void );
        void think( void );

        bool moveForward;
        bool rotateLeft;
        bool rotateRight;
        bool fireBullet;
        bool quit;
        bool fullscreen;
        bool kill;
        bool viewScore;

      private:
          InputController( void );

        void keyPressed( const SDLKey & key );
        void keyReleased( const SDLKey & key );

        SDLKey forwardKey;
        SDLKey leftKey;
        SDLKey rightKey;
        SDLKey bulletKey;
        SDLKey quitKey;
        SDLKey screenKey;
        SDLKey killKey;
        SDLKey scoreKey;

        static InputController *instance;
    };

}

#endif
```

NetworkController.cpp


```
#include "NetworkController.h"
#include "MessageMakerBase.h"
#include <iostream>
#include <string>

#include "ClientNetworkController.h"
#include "ServerNetworkController.h"

using namespace blaster;
using namespace std;


NetworkController::NetworkController( void )
{
    Connections connections = Connections(  );
    Messages messageInQueue = Messages(  );

    Messages UDPOutQueue = Messages(  );
    Messages TCPOutQueue = Messages(  );

    socketSet = SDLNet_AllocSocketSet( 32 );
    numConnections = 0;
}

// Quick fix for those without Project Builder
#if !defined (BLASTERSERVER) && !defined (BLASTERCLIENT)
#define BLASTERCLIENT
#endif

NetworkController *NetworkController::getInstance( void )
{
#ifdef BLASTERCLIENT
    return ClientNetworkController::getInstance(  );
#endif
#ifdef BLASTERSERVER
    return ServerNetworkController::getInstance(  );
#endif
}

void NetworkController::think( float time, float dt )
{
    if ( numConnections > 0 && SDLNet_CheckSockets( socketSet, 0 ) > 0 )
    {
        readMessages(  );
    }
}


void NetworkController::readMessages( void )
{
    char buffer[1024];
    TCPsocket socket;
    int id = 0;

    for ( Connections::iterator i = connections.begin(  );
          i != connections.end(  ) && !connections.empty(  ); i++ )
    {

        socket = ( ( *i ).second )->tcpsocket;
        id = ( *i ).first;

        if ( SDLNet_SocketReady( socket ) )
        {
```

NetworkController.cpp


```
            if ( SDLNet_TCP_Recv( socket, buffer, 1024 ) < 1 )
            {
                removeConnection( id );
            }
        }

    }
    // stuff that reads from tcp-socket and feeds data to message-maker
    // push messages on message queue

    // read data from UDP-socket and push on message-queue via messagemaker
    UDPpacket packet;

    NetMessage *msg = NULL;

    while ( SDLNet_UDP_Recv( udpsocket, &packet ) )
    {
        cerr << "Recived " << packet.len << " bytes of data via UDP" << endl;
        msg = MessageMakerBase::constructMessage( packet.data, packet.len );

        if ( msg )
        {
            messageInQueue.push( msg );
        }
    }

    while ( !messageInQueue.empty(  ) )
    {
        msg = messageInQueue.front(  );

        messageInQueue.pop(  );

        msg->think(  );
    }
}

void NetworkController::flush( void )
{
    UDPpacket *packet;
    NetMessage *message;

    // for each udp
    while ( !UDPOutQueue.empty(  ) )
    {
        message = UDPOutQueue.front(  );
        UDPOutQueue.pop(  );

        Uint32 messageSize = message->getSize(  );

        // create output buffer
        Uint8 *output = new Uint8[messageSize];

        message->serializeTo( output );

        // create package and put output into it

        packet = SDLNet_AllocPacket( messageSize );

        memcpy( packet->data, output, messageSize );

        packet->channel = -1;
        packet->address = getIPFromID( message->getTo(  ) );
```

```
        // we only have one udp-socket
        if ( SDLNet_UDP_Send( udpsocket, -1, packet ) == 0 )
        {
            // if errors
            removeConnection( message->getTo(  ) );
        }

        SDLNet_FreePacket( packet );

        delete message;
    }

    // for each tcp
    while ( !TCPOutQueue.empty(  ) )
    {
        message = UDPOutQueue.front(  );
        UDPOutQueue.pop(  );

        Uint32 messageSize = message->getSize(  );

        Uint8 *output = new Uint8[messageSize];

        message->serializeTo( output );

        if ( !
             ( SDLNet_TCP_Send
               ( getTCPSocketFromID( message->getTo(  ) ), output,
                 message->getSize(  ) ) < messageSize ) )
        {
            // if errors
            removeConnection( message->getTo(  ) );
        }
        delete message;
    }
}

TCPsocket NetworkController::getTCPSocketFromID( int id )
{
    Connection *con = connections[id];

    return con->tcpsocket;
}

IPaddress NetworkController::getIPFromID( int id )
{
    Connection *con = connections[id];

    return con->ip;
}


void NetworkController::pushInMessage( NetMessage * message )
{
    messageInQueue.push( message );
}

NetMessage *NetworkController::popInMessage( void )
{
    NetMessage *message = messageInQueue.front(  );

    messageInQueue.pop(  );

    return message;
```



```
}

Connection *NetworkController::addConnection( int id, TCPsocket & tcpsocket,
                                              IPaddress ip )
{
    cerr << "Adding new connection with id " << id << endl;
    Connection *con = connections[id];

    // return existing connection if exists
    if ( con )
    {
        return con;
    }

    con = new Connection( ip, tcpsocket );
    connections[id] = con;
    SDLNet_TCP_AddSocket( socketSet, tcpsocket );
    numConnections++;

    cerr << "numConnections: " << numConnections << endl;

    return con;
}


void NetworkController::removeConnection( int id )
{
    Connection *con = connections[id];

    if ( con )
    {
        cerr << "Breaking connection with id: " << id << endl;
        SDLNet_TCP_DelSocket( socketSet, con->tcpsocket );
        Connections::iterator i = connections.find( id );
        connections.erase( i );
        numConnections--;
        delete con;

        cerr << "numConnections: " << numConnections << endl;
    }
}


void NetworkController::UDPSendMessage( NetMessage * message )
{
    UDPOutQueue.push( message );
}

void NetworkController::TCPSendMessage( NetMessage * message )
{
    TCPOutQueue.push( message );
}
```

NetworkController.h Page 1/2

```
#ifndef NETWORKCONTROLLER_H
#define NETWORKCONTROLLER_H

#include "SDL_net.h"
#include "NetworkController.h"
#include "NetMessage.h"
#include "Connection.h"

#include <queue>
#include <map>

using namespace std;

namespace blaster
{
    typedef std::queue < NetMessage * >Messages;
    typedef std::map < int, Connection * >Connections;

    class NetworkController
    {

      public:
        virtual void think( float time, float dt );

        static NetworkController *getInstance( void );
          virtual ~ NetworkController( void )
        {
        }

        void flush( void );
        NetMessage *popMessage( void );

        void TCPSendMessage( NetMessage * message );
        void UDPSendMessage( NetMessage * message );

        // Dummy methods that make sure that both our network-controllers
        // have the same interface
        virtual void startServer( Uint16 port )
        {
        }
        virtual bool connect( const std::string & address, Uint16 port )
        {
            return false;
        }

        virtual void disconnect( void ) {}

        int numConnections;

      private:
        void readMessages( void );

      protected:
        NetworkController( void );
        NetMessage *parseMessage( char *data, int size );

        TCPsocket getTCPSocketFromID( int id );
        IPaddress getIPFromID( int id );

        NetMessage *popInMessage( void );
        void pushMessageOnQueue( NetMessage * message );
        void pushInMessage( NetMessage * message );
```

NetworkController.h Page 2/2

```
        // *** variables ***

        // used for checking udp/tcp sockets for activity
        SDLNet_SocketSet socketSet;

        // Creates new tcp connection to ip
        Connection *addConnection( int id, TCPsocket & tcpsocket,
                                   IPaddress ip );

        // removes and disconnects an tcp-connection
        virtual void removeConnection( int id );

        // UDPis connectionsless so theres no need to have more than one socket
        UDPsocket udpsocket;

        // map of connections
        Connections connections;
        // queue of incomming messages
        Messages messageInQueue;
        // queue of outgoing messages
        Messages UDPOutQueue;
        Messages TCPOutQueue;
    };
}

#endif
```

ServerController.cpp

Page 1/2

```
#include "ServerController.h"
#include "SDL.h"
#include "SDL_net.h"
#include "NetworkController.h"
#include "GameController.h"
#include <iostream>

using namespace blaster;
using namespace std;

// happy now gcc ?
ServerController *ServerController::instance = NULL;

ServerController::ServerController( void )
{
    running = true;

    setupSDL(  );
    setupSDLNet(  );
}

void ServerController::setupSDL(  )
{
    if ( SDL_Init( SDL_INIT_TIMER ) < 0 )
    {
        fprintf( stderr, "Couldn't initialize SDL: %s\n", SDL_GetError(  ) );
        exit( 1 );
    }
}

void ServerController::setupSDLNet( void )
{
    // initialize SDL_net
    if ( SDLNet_Init(  ) == -1 )
    {
        printf( "SDLNet_Init: %s\n", SDLNet_GetError(  ) );
        exit( 2 );
    }
}

void ServerController::run( void )
{
    long launchtime = SDL_GetTicks(  );
    long oldtime = 0;
    long newtime = 0;

    float time = 0.0f;
    float dt = 0.0f;

    NetworkController *network = NetworkController::getInstance(  );

    network->startServer( 1337 );

    while ( running )
    {
        newtime = SDL_GetTicks(  ) - launchtime;
        dt = ( float ) ( newtime - oldtime ) * 0.001f;
        time = ( float ) newtime *0.001f;

        oldtime = newtime;

        /*NetworkController * */ network = NetworkController::getInstance(  );
```

ServerController.cpp

Page 2/2

```
        network->think( time, dt );

        // Update the game world
        GameController::getInstance(  )->think( time, dt );

        network->flush(  );
    }

    // shutdown SDL_net
    SDLNet_Quit(  );
    SDL_Quit(  );
}


ServerController *ServerController::getInstance( void )
{
    if ( instance == NULL )
    {
        instance = new ServerController(  );
    }

    return instance;
}



int main( int argc, char **argv )
{
    cout << "Starting ServerController" << endl;
    ServerController *server;

    server = ServerController::getInstance(  );
    server->run(  );

    return ( 0 );
}
```

ServerController.h

Page 1/1

```
#ifndef SERVERCONTROLLER_H
#define SERVERCONTROLLER_H

#include "ServerNetworkController.h"

namespace blaster
{
    class ServerController
    {
      public:
        ~ServerController( void );
        static ServerController *getInstance( void );
        void think( void );
        void run( void );
        void quit( void )
        {
            running = false;
        }

      private:
        //vars
        static ServerController *instance;

          ServerController( void );
        void setupSDL( void );
        void setupSDLNet( void );

        bool running;
    };
}

#endif
```

ServerNetworkController.cpp

Page 1/2

```
#include "ServerNetworkController.h"
#include <iostream>
#include <string>

using namespace blaster;
using namespace std;

ServerNetworkController *ServerNetworkController::instance = NULL;


ServerNetworkController::ServerNetworkController( void ):NetworkController(  )
{
    nextID = 1;
    nextClientCheck = 0;
}

void ServerNetworkController::think( float time, float dt )
{
    // accept new connections
    if ( time > nextClientCheck )
    {
        checkForNewClients(  );

        nextClientCheck += 0.1f;

        if ( nextClientCheck < time )
            nextClientCheck = time;
    }

    // get all incomming stuff
    NetworkController::think( time, dt );
}


void ServerNetworkController::checkForNewClients( void )
{
    IPaddress *remoteip;
    TCPsocket tcpclient;
    Uint32 ipaddr;

    // try to accept a connection
    tcpclient = SDLNet_TCP_Accept( tcpServerSocket );

    // no connection-attempts
    if ( !tcpclient )
        return;

    cerr << "new connection detected..." << endl;

    // get the clients IP and port number
    remoteip = SDLNet_TCP_GetPeerAddress( tcpclient );
    if ( !remoteip )
    {
        cerr << "SDLNet_TCP_GetPeerAddress: " << SDLNet_GetError(  ) << endl;;
        return;
    }


    // print out the clients IP and port number
    ipaddr = SDL_SwapBE32( remoteip->host );
    cerr << "Accepted a connection from "
        << ( ipaddr >> 24 ) << "."
        << ( ( ipaddr >> 16 ) & 0xff ) << "."
```

ServerNetworkController.cpp


```
        << ( ( ipaddr >> 8 ) & 0xff ) << "."
        << ( ipaddr & 0xff ) << " on port " << ( remoteip->port ) << endl;

    // add connection to io/queue
    addConnection( nextID, tcpclient, *remoteip );
    nextID++;
}


void ServerNetworkController::startServer( Uint16 port )
{
    IPaddress ip;

    // Resolve the argument into an IPaddress type
    if ( SDLNet_ResolveHost( &ip, NULL, port ) == -1 )
    {
        cerr << "Could not resolve localhost" << SDLNet_GetError(  ) << endl;
        exit( 3 );
    }

    cerr << "Resolved host" << endl;

    // open the server socket
    tcpServerSocket = SDLNet_TCP_Open( &ip );
    if ( !tcpServerSocket )
    {
        cerr << "Could not open TCP-server-socket: " << SDLNet_GetError(  ) <<
            endl;
        exit( 4 );
    }

    cerr << "TCP server socket created" << endl;

    // open udp-socket
    udpsocket = SDLNet_UDP_Open( port );
    if ( !udpsocket )
    {
        cerr << "SDLNet_UDP_Open: " << SDLNet_GetError(  ) << endl;
        exit( 2 );
    }

    cerr << "UDP socket opened" << endl;
}



ServerNetworkController *ServerNetworkController::getInstance( void )
{
    if ( instance == NULL )
    {
        instance = new ServerNetworkController(  );
    }

    return instance;
}
```

ServerNetworkController.h


```
#ifndef SERVERNETWORKCONTROLLER_H
#define SERVERNETWORKCONTROLLER_H

#include "SDL.h"
#include "SDL_net.h"
#include <deque>
#include "ServerNetworkController.h"
#include "NetworkController.h"

namespace blaster
{
    class ServerNetworkController:public NetworkController
    {
      public:
        virtual void think( float time, float dt );
        virtual void startServer( Uint16 port );

        static ServerNetworkController *getInstance( void );
          virtual ~ ServerNetworkController( void )
        {
        }

      private:
        //vars

          TCPsocket tcpServerSocket;
        static ServerNetworkController *instance;
        int port;
        int nextID;

        //functions
          ServerNetworkController( void );

        void checkForNewClients( void );

        float nextClientCheck;
    };
}

#endif
```

Font.cpp


```
#include "Font.h"
#include "SDL_endian.h"
#include <fstream>

// Font constructor. Read and upload the texture containing the font
Font::Font( const std::string & filename )
{
    fontname = filename;
}

void Font::reload( void )
{
    loadFont( fontname );
}

void Font::loadFont( const std::string & filename )
{
    Uint32 tmp = 0;
    Sint32 i;

    std::ifstream file;

    file.open( filename.c_str(  ), std::ios::binary );

    if ( !file.is_open(  ) )
        throw "Failed to open font\n";

    // The first 4 bytes should contain the number '6666' if this is a valid fon
t file.
    file.read( ( char * ) &tmp, sizeof( tmp ) );
    tmp = SDL_SwapLE32( tmp );  // Since we're working with values larger than o
ne byte, we need to handle byte-endianness

    if ( tmp != 6666 )
        throw "Wrong font format\n";

    file.read( ( char * ) &tmp, sizeof( tmp ) );
    textureWidth = SDL_SwapLE32( tmp );

    file.read( ( char * ) &tmp, sizeof( tmp ) );
    textureHeight = SDL_SwapLE32( tmp );

    file.read( ( char * ) &tmp, sizeof( tmp ) );
    fontHeight = SDL_SwapLE32( tmp );

    union
    {
        float f;
        Uint32 i;
    }
    converter;

    const GLfloat fTexWidth = ( float ) textureWidth;
    const GLfloat fTexHeight = ( float ) textureHeight;

    // Read each character from the bitmap
    for ( i = 0; i < 256; i++ )
    {
        // Read top v-coord
        file.read( ( char * ) &tmp, sizeof( tmp ) );
        chars[i].texCoords[1] = ( float ) SDL_SwapLE32( tmp ) / fTexHeight;

        // Read left u-coord
```

Font.cpp


```
        file.read( ( char * ) &tmp, sizeof( tmp ) );
        chars[i].texCoords[0] = ( float ) SDL_SwapLE32( tmp ) / fTexWidth;

        // Read bottom v-coord
        file.read( ( char * ) &tmp, sizeof( tmp ) );
        chars[i].texCoords[3] = ( float ) SDL_SwapLE32( tmp ) / fTexHeight;

        // Read bottom u-coord
        file.read( ( char * ) &tmp, sizeof( tmp ) );
        chars[i].texCoords[2] = ( float ) SDL_SwapLE32( tmp ) / fTexWidth;

        // Is this character valid? (if not, a space will be displayed instead)
        file.read( ( char * ) &tmp, sizeof( tmp ) );
        chars[i].enabled = ( SDL_SwapLE32( tmp ) != 0 );

        // Read the character width
        file.read( ( char * ) &tmp, sizeof( tmp ) );
        converter.i = SDL_SwapLE32( tmp );
        chars[i].width = converter.f * fontHeight;
    }

    if ( ( file.eof(  ) ) || ( file.fail(  ) ) )
        throw "Font::loadFont - error reading from file";

    // Allocate two buffers to store the actual bitmap information before upload
ing it as a texture

    char *buffer, *buffer2;
    const int bufferSize = textureWidth * textureHeight;

    buffer = ( char * ) malloc( bufferSize );

    if ( buffer == 0 )
        throw "Unable to allocate memory";

    file.read( buffer, bufferSize );

    // We're done reading from the file
    file.close(  );

    // Allocate a buffer large enough to store both pixel data and alpha
    buffer2 = ( char * ) malloc( bufferSize * 2 );

    if ( buffer2 == NULL )
        throw "Unable to allocate memory";

    for ( i = 0; i < bufferSize; i++ )
    {
        buffer2[i * 2] = buffer[i];
        buffer2[i * 2 + 1] = buffer[i];
    }

    // Upload the bitmap as a texture
    glGenTextures( 1, &textureId );
    glBindTexture( GL_TEXTURE_2D, textureId );
    glTexImage2D( GL_TEXTURE_2D, 0, GL_LUMINANCE_ALPHA, textureWidth,
                  textureHeight, 0, GL_LUMINANCE_ALPHA, GL_UNSIGNED_BYTE,
                  buffer2 );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP );
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP );
    glBindTexture( GL_TEXTURE_2D, 0 );
```

Font.cpp


```
    free( buffer );
    free( buffer2 );
}

void Font::draw( const std::string & value )
{
    // Enable texturemapping
    glEnable( GL_TEXTURE_2D );

    // Select the texture associated with this font
    glBindTexture( GL_TEXTURE_2D, textureId );

    char c;                     // holds the current character in the string
    GLfloat x = 0.0f;

    // Draw each character in the string as a textured rectangle (two triangles)
    for ( Uint32 i = 0; i < value.length(  ); i++ )
    {
        c = value[i];

        // Check if the character is represented in the texture
        if ( !chars[c].enabled )
            c = 32;             // replace with the space character

        glBegin( GL_TRIANGLE_STRIP );

        // Top left corner of the current character
        glTexCoord2f( chars[c].texCoords[0], chars[c].texCoords[1] );
        glVertex2f( x, 0.0f );

        // Bottom left
        glTexCoord2f( chars[c].texCoords[0], chars[c].texCoords[3] );
        glVertex2f( x, fontHeight );

        // Move to the right according to the character width
        x += chars[c].width;

        // Top right
        glTexCoord2f( chars[c].texCoords[2], chars[c].texCoords[1] );
        glVertex2f( x, 0.0f );

        // Bottom right
        glTexCoord2f( chars[c].texCoords[2], chars[c].texCoords[3] );
        glVertex2f( x, fontHeight );

        glEnd(  );
    }

    glDisable( GL_TEXTURE_2D );
}

//Font destructor
Font::~Font( void )
{
    glDeleteTextures( 1, &textureId );
}
```

Font.h


```
#ifndef FONT_H
#define FONT_H

#include "SDL_opengl.h"
#include <string>

class Character
{
  public:
    GLfloat texCoords[4];
    GLfloat width;

    bool enabled;
};

class Font
{
  public:
    Font( const std::string & filename );
     ~Font( void );

    void draw( const std::string & value );

    GLfloat getFontHeight( void )
    {
        return fontHeight;
    }

    void reload( void );

  private:
    void loadFont( const std::string & filename );

    GLfloat fontHeight;
    Character chars[256];
    GLuint textureWidth;
    GLuint textureHeight;
    GLuint textureId;

    std::string fontname;
};

#endif // FONT_H
```

GameView.cpp

Page 1/1

```
#include "ClientController.h"
#include "GameView.h"
#include "GameController.h"
#include "Level.h"
#include "Player.h"

using namespace blaster;

GameView::GameView( View * parent ):View( parent ), viewPosition( 0, 0 )
{
}

void GameView::drawContent( void )
{
    if ( parent )
        parent->drawContent(  );

    Level *level = Level::getInstance(  );
    Map *map = level->getMap(  );

    Objects *ships = level->getShips(  );
    Objects *bullets = level->getBullets(  );

    Uint32 localClient =
        ClientController::getInstance(  )->getLocalPlayerID(  );

    Ship *ship =
        GameController::getInstance(  )->getPlayer( localClient )->
        getShip(  );

    glMatrixMode( GL_PROJECTION );

    glPushMatrix(  );

    if ( ship != NULL )
    {
        viewPosition = ship->getPosition(  );
    }

    glTranslatef( -viewPosition.x, -viewPosition.y, 0.0f );

    glMatrixMode( GL_MODELVIEW );

    map->draw(  );

    for ( Objects::iterator i = bullets->begin(  ); i != bullets->end(  );
          i++ )
    {
        ( *i )->draw(  );
    }

    for ( Objects::iterator i = ships->begin(  ); i != ships->end(  ); i++ )
    {
        ( *i )->draw(  );
    }

    glMatrixMode( GL_PROJECTION );

    glPopMatrix(  );
}
```

GameView.h

Page 1/1

```
#ifndef GAMEVIEW_H
#define GAMEVIEW_H

#include "View.h"
#include "Vector.h"

namespace blaster
{

    class GameView:public View
    {
      public:
        GameView( View * parent );
        virtual void drawContent( void );
      private:
          Vector viewPosition;
    };

}

#endif
```

ScoreView.cpp


```
#include "ClientController.h"
#include "ScoreView.h"
#include "InputController.h"
#include "GameController.h"
#include "Player.h"

#include <iostream>

using namespace blaster;
using namespace std;

ScoreView::ScoreView( View * parent ):View( parent )
{
    // get this from somewhere else
    screenWidth = 640;
    screenHeight = 480;

    f = new Font( "verdana.fnt" );
}

void ScoreView::drawContent( void )
{
    if ( parent )
        parent->drawContent(  );

    if ( !InputController::getInstance(  )->viewScore )
    {
        return;
    }

    glMatrixMode( GL_PROJECTION );

    glPushMatrix(  );

    glLoadIdentity(  );

    /* left, right, bottom, top... */
    glOrtho( 0.0f, screenWidth, screenHeight, 0.0f, -1.0f, 1.0f );

    glMatrixMode( GL_MODELVIEW );

    glPushMatrix(  );

    glLoadIdentity(  );

    // calculate box size

    float x1 = screenWidth / 6;
    float y1 = screenHeight / 6;

    float boxWidth = screenWidth - ( screenWidth / 6 ) * 2;
    float x2 = x1 + boxWidth;
    float y2 = screenHeight - ( screenHeight / 6 );

    glColor4f( 0.5, 0.5, 0.8, 0.5 );
    glBegin( GL_QUADS );
    glVertex2f( x1, y1 );
    glVertex2f( x2, y1 );
    glVertex2f( x2, y2 );
    glVertex2f( x1, y2 );
    glEnd(  );

    glTranslatef( x1 + 10, y1 + 10, 0.0 );
```

ScoreView.cpp


```
    string playername;
    char score[10];

    Ship *ship;

    Uint32 localClient =
        ClientController::getInstance(  )->getLocalPlayerID(  );

    Player *player = GameController::getInstance(  )->getPlayer( localClient );
// localid

    glColor3f( 1.0, 1.0, 1.0 );

    // insert magical forloop that gets all players and for each does
    // start loop
    ship = player->getShip(  );
    if ( ship )
    {
        playername = player->getName(  );
        glTranslatef( 0, f->getFontHeight(  ) + 10, 0.0 );
        f->draw( playername.c_str(  ) );

        // right collumn
        sprintf( score, "%d", player->getScore(  ) );
        glTranslatef( boxWidth - 100, 0, 0.0 );
        f->draw( score );

        // back again
        glTranslatef( -( boxWidth - 100 ), 0, 0.0 );
    }
    // end loop

    glPopMatrix(  );

    glMatrixMode( GL_PROJECTION );
    glPopMatrix(  );

    glMatrixMode( GL_PROJECTION );
    glPopMatrix(  );
}

void ScoreView::init( void )
{
    if ( parent )
        parent->init(  );

    f->reload(  );
}
```

ScoreView.h

Page 1/1

```
#ifndef SCOREVIEW_H
#define SCOREVIEW_H

#include "View.h"
#include "Font.h"

namespace blaster
{

    class ScoreView:public View
    {
      public:
        ScoreView( View * parent );
        virtual void drawContent( void );
        void init( void );

      private:
          Font * f;
        int screenWidth;
        int screenHeight;
    };

}

#endif
```

StatusBar.cpp

Page 1/2

```
#include "StatusBar.h"
#include "SDL.h"
#include "SDL_opengl.h"
#include <iostream>

using namespace blaster;
using namespace std;

StatusBar::StatusBar( float width, float height )
{
    colorGood = Vector( 0, 1, 0 );
    colorBad = Vector( 1, 0, 0 );

    this->width = width;
    this->height = height;
}

void StatusBar::draw( float max, float current )
{
    float goodness = ( current / max );

    Vector color = colorGood * goodness + colorBad * ( 1 - goodness );

    // draw bar
    glColor3f( color.x, color.y, color.z );
    // padding

    float innerpad = 4;

    float x1 = 0;
    float y1 = 0;

    float x2 = width - innerpad * 2;
    float y2 = height - innerpad * 2;

    x2 = x2 * goodness;

    x1 = x1 + innerpad;
    y1 = y1 + innerpad;

    x2 = x2 + innerpad;
    y2 = y2 + innerpad;


    glBegin( GL_QUADS );
    glVertex2f( x1, y1 );
    glVertex2f( x2, y1 );
    glVertex2f( x2, y2 );
    glVertex2f( x1, y2 );
    glEnd(  );

    // draw box around bar
    glColor3f( 1.0, 1.0, 1.0 );
    glBegin( GL_LINES );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( width, 0.0 );

    glVertex2f( width, 0.0 );
    glVertex2f( width, height );

    glVertex2f( width, height );
    glVertex2f( 0.0, height );
```

StatusBar.cpp

Page 2/2

```
    glVertex2f( 0.0, height );
    glVertex2f( 0.0, 0.0 );
    glEnd(  );

}
```

StatusBar.h

Page 1/1

```
#ifndef STATUSBAR_H
#define STATUSBAR_H

#include "Vector.h"

namespace blaster
{

    class StatusBar
    {
      public:
        StatusBar( float width, float height );

        void draw( float max, float value );


      private:
        float width;
        float height;

        Vector colorGood;
        Vector colorBad;
    };

}

#endif
```

Statusview.cpp

Page 1/3

```
#include "ClientController.h"
#include "StatusView.h"
#include "Ship.h"
#include "Player.h"
#include "GameController.h"

#include <iostream>

using namespace blaster;
using namespace std;

StatusView::StatusView( View * parent ):View( parent )
{
    screenWidth = 640;
    screenHeight = 480;

    barWidth = 100.0;
    barHeight = 20.0;

    // padding on left/right for each bar
    barXpad = ( screenWidth / 4 - barWidth ) / 2;

    ammoBar = new StatusBar( barWidth, barHeight );
    fuelBar = new StatusBar( barWidth, barHeight );
    energyBar = new StatusBar( barWidth, barHeight );

    f = new Font( "verdana.fnt" );
    f->reload(  );
}

void StatusView::drawContent( void )
{
    if ( parent )
        parent->drawContent(  );

    Uint32 localClient =
        ClientController::getInstance(  )->getLocalPlayerID(  );

    Player *player = GameController::getInstance(  )->getPlayer( localClient );

    Ship *ship = player->getShip(  );

    // can't draw anything without a ship
    if ( !ship )
        return;

    glMatrixMode( GL_PROJECTION );

    glPushMatrix(  );

    glLoadIdentity(  );

    /* left, right, bottom, top... */
    glOrtho( 0.0f, screenWidth, screenHeight, 0.0f, -1.0f, 1.0f );


    glMatrixMode( GL_MODELVIEW );

    glPushMatrix(  );

    glLoadIdentity(  );
```

Statusview.cpp

Page 2/3

```
    // draw bars

    // scorebox
    glTranslatef( barXpad, screenHeight - ( barHeight + 5 ), 0.0f );
    glColor3f( 0.0, 0.7, 0.0 );
    glBegin( GL_LINES );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( barWidth, 0.0 );

    glVertex2f( barWidth, 0.0 );
    glVertex2f( barWidth, barHeight );

    glVertex2f( barWidth, barHeight );
    glVertex2f( 0.0, barHeight );

    glVertex2f( 0.0, barHeight );
    glVertex2f( 0.0, 0.0 );
    glEnd(  );


    glTranslatef( barWidth + 2 * barXpad, 0, 0.0f );
    ammoBar->draw( ship->ammoCapacity(  ), ship->getAmmo(  ) );


    glTranslatef( barWidth + 2 * barXpad, 0, 0.0f );
    fuelBar->draw( ship->fuelCapacity(  ), ship->getFuel(  ) );

    glTranslatef( barWidth + 2 * barXpad, 0, 0.0f );
    energyBar->draw( ship->energyCapacity(  ), ship->getEnergy(  ) );

    // draw strings
    glLoadIdentity(  );
    glColor3f( 1.0, 1.0, 1.0 );

    char buf[256];

    glTranslatef( barXpad,
                  screenHeight - ( barHeight + 5 ) - f->getFontHeight(  ),
                  0.0f );
    glTranslatef( barWidth + 2 * barXpad, 0, 0.0f );
    sprintf( buf, "Ammo: %d", ship->getAmmo(  ) );
    f->draw( buf );

    glTranslatef( barWidth + 2 * barXpad, 0, 0.0f );
    sprintf( buf, "Fuel: %f", ship->getFuel(  ) / 1 );
    f->draw( buf );

    glTranslatef( barWidth + 2 * barXpad, 0, 0.0f );
    sprintf( buf, "Energy: %f", ship->getEnergy(  ) / 1 );
    f->draw( buf );


    glLoadIdentity(  );
    glColor3f( 1.0, 1.0, 1.0 );
    glTranslatef( barXpad + 2, screenHeight - ( barHeight + 3 ), 0.0f );
    sprintf( buf, "Score: %d", player->getScore(  ) );
    f->draw( buf );


    glPopMatrix(  );

    glMatrixMode( GL_PROJECTION );
```

Statusview.cpp


```
    glPopMatrix(  );

    glMatrixMode( GL_PROJECTION );

    glPopMatrix(  );

}


void StatusView::init( void )
{
    if ( parent )
        parent->init(  );

    f->reload(  );
}
```

Statusview.h


```
#ifndef STATUS_H
#define STATUS_H

#include "View.h"
#include "Font.h"
#include "Vector.h"
#include "StatusBar.h"

namespace blaster
{

    class StatusView:public View
    {
      public:
        StatusView( View * parent );
        virtual void drawContent( void );
        void init( void );

      private:
          Font * f;
        int screenWidth;
        int screenHeight;
        float barWidth;
        float barHeight;
        float barXpad;

        void drawBar( float width, float height, float cur, float max );

        StatusBar *ammoBar;
        StatusBar *fuelBar;
        StatusBar *energyBar;   // crunchy
    };

}

#endif
```

View.cpp

Page 1/1

```
#include "View.h"

using namespace blaster;

View::View( View * p )
{
    parent = p;
}

View::~View( void )
{
    delete parent;
}

void View::drawFrame( void )
{
    glClear( GL_COLOR_BUFFER_BIT );

    drawContent(  );

    glFlush(  );

    SDL_GL_SwapBuffers(  );
}

void View::init( void )
{
    if ( parent )
        parent->init(  );
}
```

View.h

Page 1/1

```
#ifndef VIEW_H
#define VIEW_H

#include "SDL.h"
#include "SDL_opengl.h"

namespace blaster
{

    class View
    {
      public:
        View( View * p );
        virtual ~ View( void );

        virtual void init( void );
        virtual void drawFrame( void );
        virtual void drawContent( void ) = 0;

      protected:
          View * parent;
    };

}

#endif                          // VIEW_H
```

Matrix.cpp — Page 1/3

```
#include "Matrix.h"

#include "Vector.h"

#include "Math.h"
#include "String.h"

#include <iostream>

Matrix::Matrix( void )
{
    m = new float[16];

    loadIdentity(  );
}

Matrix::Matrix( const Matrix & n )
{
    m = new float[16];

    memcpy( m, n.m, sizeof( float ) * 16 );
}

Matrix::~Matrix( void )
{
    delete m;
}

void Matrix::loadIdentity( void )
{
    m[0] = 1.0f;
    m[4] = 0.0f;
    m[8] = 0.0f;
    m[12] = 0.0f;
    m[1] = 0.0f;
    m[5] = 1.0f;
    m[9] = 0.0f;
    m[13] = 0.0f;
    m[2] = 0.0f;
    m[6] = 0.0f;
    m[10] = 1.0f;
    m[14] = 0.0f;
    m[3] = 0.0f;
    m[7] = 0.0f;
    m[11] = 0.0f;
    m[15] = 1.0f;
}

Matrix Matrix::translateMatrix( float a, float b, float c )
{
    Matrix mat;

    mat.m[12] = a;
    mat.m[13] = b;
    mat.m[14] = c;

    return mat;
}

Matrix Matrix::scaleMatrix( float a, float b, float c )
{
    Matrix mat;
```

Matrix.cpp — Page 2/3

```
    mat.m[0] = a;
    mat.m[5] = b;
    mat.m[10] = c;

    return mat;
}

Matrix Matrix::rotateZMatrix( float angle )
{
    Matrix mat;

    const float cosa = cos( angle );
    const float sina = sin( angle );

    mat.m[0] = cosa;
    mat.m[4] = -sina;
    mat.m[1] = sina;
    mat.m[5] = cosa;

    return mat;
}

Matrix & operator*=( Matrix & left, const Matrix & right )
{
    float *p = left.m;
    float *q = right.m;

    left.m = new float[16];

    for ( int j = 0; j < 4; j++ )
    {
        for ( int i = 0; i < 4; i++ )
        {
            left.m[i + 4 * j] = p[0 + i] * q[4 * j + 0] +
                p[4 + i] * q[4 * j + 1] +
                p[8 + i] * q[4 * j + 2] + p[12 + i] * q[4 * j + 3];
        }
    }

    delete p;

    return left;
}

Vector operator*( Matrix & matrix, const Vector & v )
{
    float a =
        v.x * matrix.m[0] + v.y * matrix.m[4] + v.z * matrix.m[8] +
        1 * matrix.m[12];
    float b =
        v.x * matrix.m[1] + v.y * matrix.m[5] + v.z * matrix.m[9] +
        1 * matrix.m[13];
    float c =
        v.x * matrix.m[2] + v.y * matrix.m[6] + v.z * matrix.m[10] +
        1 * matrix.m[14];

    return Vector( a, b, c );
}

std::ostream & operator<<( std::ostream & out, const Matrix & m )
{
    for ( int j = 0; j < 4; j++ )
    {
```

Matrix.cpp


```
        out << "[";

        for ( int i = 0; i < 4; i++ )
        {
            out << "\t" << m.m[i + 4 * j];
        }

        out << "\t]" << std::endl;
    }

    return out;
}
```

Matrix.h


```
#ifndef MATRIX_H
#define MATRIX_H

#include <iostream>

class Vector;

class Matrix
{
  public:
    Matrix( void );
    Matrix( const Matrix & n );
     ~Matrix( void );

    void loadIdentity( void );

    static Matrix translateMatrix( float a, float b, float c );
    static Matrix scaleMatrix( float a, float b, float c );
    static Matrix rotateZMatrix( float angle );


    float *m;
};

Vector operator*( Matrix & m, const Vector & v );
Matrix & operator*=( Matrix & left, const Matrix & right );

std::ostream & operator<<( std::ostream & out, const Matrix & m );

#endif // MATRIX_H
```

Vector.cpp


```
#include "Vector.h"
#include "Matrix.h"
#include <iostream>

using namespace std;

const Vector Vector::ZERO( 0.0f, 0.0f, 0.0f );

const Vector Vector::operator+( const Vector & v ) const
{
    return Vector( x + v.x, y + v.y, z + v.z );
}

const Vector Vector::operator-( const Vector & v ) const
{
    return Vector( x - v.x, y - v.y, z - v.z );
}

const float Vector::operator*( const Vector & v ) const
{
    return x * v.x + y * v.y + z * v.z;
}

const Vector Vector::operator*( const float value ) const
{
    return Vector( x * value, y * value, z * value );
}

const Vector Vector::operator/( const float value ) const
{
    return Vector( x / value, y / value, z / value );
}

const Vector Vector::cross( const Vector & v ) const
{
    return Vector( 0, 0, 0 );
}

void Vector::transform( Matrix & matrix )
{
    Vector v = matrix * ( *this );

    x = v.x;
    y = v.y;
    z = v.z;
}

Vector & operator+=( Vector & left, const Vector & right )
{
    left.x += right.x;
    left.y += right.y;
    left.z += right.z;

    return left;
}

Vector & operator*=( Vector & left, const float v )
{
    left.x *= v;
    left.y *= v;
    left.z *= v;

    return left;
```

Vector.cpp


```
}

std::ostream & operator<<( std::ostream & out, const Vector & v )
{
    out << "(" << v.x << "," << v.y << "," << v.z << ")";

    return out;
}
```

```
#ifndef VECTOR_H
#define VECTOR_H

#include "stdio.h"
#include "math.h"

#include <iostream>

class Matrix;

class Vector
{
  public:
    Vector( void ):x( 0.0f ), y( 0.0f ), z( 0.0f )
    {
    }
    Vector( float x, float y, float z = 0.0f ):x( x ), y( y ), z( z )
    {
    }
    Vector( const Vector & v ):x( v.x ), y( v.y ), z( v.z )
    {
    }

    const Vector operator+( const Vector & v ) const;
    const Vector operator-( const Vector & v ) const;
    const float operator*( const Vector & v ) const;
    const Vector operator*( const float value ) const;
    const Vector operator/( const float value ) const;

//    inline float dot( Vector &rvalue );
    const Vector cross( const Vector & rvalue ) const;

    void transform( Matrix & matrix );

    const float length( void ) const
    {
        return sqrt( x * x + y * y + z * z );
    }
    const Vector normalize( void ) const
    {
        return *this / length(  );
    }

    static const Vector ZERO;

    float x;
    float y;
    float z;
};

std::ostream & operator<<( std::ostream & out, const Vector & v );
Vector & operator+=( Vector & left, const Vector & right );
Vector & operator*=( Vector & left, const Vector & v );

#endif
```

Appendiks D

Editor kilde kode

Exporter.java


```
/**
 * ------------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------------
 */

package dk.ruc.blaster.export;

import java.awt.Dimension;
import java.io.File;
import java.io.IOException;
import java.util.Iterator;

import javax.swing.JOptionPane;

import dk.ruc.blaster.model.Map;
import dk.ruc.blaster.model.Triangle;
import dk.ruc.blaster.model.Vector;
import dk.ruc.blaster.model.Vertex;
import dk.ruc.blaster.ui.BlasterEditor;
import electric.xml.Document;
import electric.xml.Element;

/**
 * Static class to export a Map to a xml file
 *
 * Last change by: $Author: dyregod $
 *
 * $Header: /var/cvs/Alwazah\040Editor/source/dk/ruc/blaster/export/Exporter.jav
a,v 1.15 2002/12/16 00:19:08 dyregod Exp $
 *
 * @version $Revision: 1.15 $
 */
public class Exporter
{
    /**
     * Export method. Does the actual map -> xml conversion
     * @param map the map file to export
     * @param file the xml file to be written to
     */
    public static void export(Map map, File file)
    {

        // create empty document
        Document document = new Document();

        // create root
        Element mapElement = document.setRoot("map");
        mapElement.setAttribute("width", String.valueOf(map.width));
        mapElement.setAttribute("height", String.valueOf(map.height));
        mapElement.setAttribute("gravityx", String.valueOf(map.gravityx));
        mapElement.setAttribute("gravityy", String.valueOf(map.gravityy));
        mapElement.setAttribute("name", map.name);

        // add points
        Element points = mapElement.addElement("points");
        points.setAttribute("size", String.valueOf(map.vertices.size()));

        Dimension d = map.getDimensions();
```

Exporter.java


```
        for (Iterator iter = map.vertices.iterator(); iter.hasNext();)
        {
            Vertex v = (Vertex) iter.next();
            Vector p = v.getPosition();

            Element point = points.addElement("point");
            point.setAttribute("x", String.valueOf(p.x - d.width / 2));
            point.setAttribute("y", String.valueOf(p.y - d.height / 2));
            point.setAttribute("id", String.valueOf(map.vertices.indexOf(v)));
        }

        Element polygons = mapElement.addElement("polygons");
        polygons.setAttribute("size", String.valueOf(map.triangles.size()));

        for (Iterator iter = map.triangles.iterator(); iter.hasNext();)
        {
            Triangle triangle = (Triangle) iter.next();

            Element polygon = polygons.addElement("polygon");
            polygon.setAttribute(
                "id",
                String.valueOf(map.triangles.indexOf(triangle)));

            // add color element
            Element color = polygon.addElement("color");
            color.setAttribute(
                "r",
                String.valueOf(triangle.getColor().getRed() / 255.0f));
            color.setAttribute(
                "g",
                String.valueOf(triangle.getColor().getGreen() / 255.0f));
            color.setAttribute(
                "b",
                String.valueOf(triangle.getColor().getBlue() / 255.0f));

            Element edgeElement;
            //  Edge edge;
            // add (hopefully) 3 edges
            int[] v = new int[3];
            Vertex ver = triangle.vertices[2];
            Vector vv = ver.getPosition();
            float dot =
                vv.subtract(triangle.edges[0].getB().getPosition()).dot(
                    triangle.edges[0].normal());
            if (dot < 0)
            {
                v[0] = map.vertices.indexOf(triangle.edges[0].getA());
                v[1] = map.vertices.indexOf(triangle.edges[0].getB());
                v[2] = map.vertices.indexOf(ver);
                edgeElement = polygon.addElement("edge");
                edgeElement.setAttribute("p1", String.valueOf(v[0]));
                edgeElement.setAttribute(
                    "platform",
                    triangle.edges[0].getEdgeType() ? "true" : "false");
                edgeElement.setAttribute("id", "0");
                edgeElement = polygon.addElement("edge");
                edgeElement.setAttribute("p1", String.valueOf(v[1]));
                edgeElement.setAttribute(
                    "platform",
                    triangle.edges[1].getEdgeType() ? "true" : "false");
                edgeElement.setAttribute("id", "1");
                edgeElement = polygon.addElement("edge");
```

Exporter.java


```
                edgeElement.setAttribute("p1", String.valueOf(v[2]));
                edgeElement.setAttribute(
                    "platform",
                    triangle.edges[2].getEdgeType() ? "true" : "false");
                edgeElement.setAttribute("id", "2");
            }
            else
            {
                v[0] = map.vertices.indexOf(triangle.edges[0].getB());
                v[1] = map.vertices.indexOf(ver);
                v[2] = map.vertices.indexOf(triangle.edges[0].getA());
                edgeElement = polygon.addElement("edge");
                edgeElement.setAttribute("p1", String.valueOf(v[2]));
                edgeElement.setAttribute(
                    "platform",
                    triangle.edges[0].getEdgeType() ? "true" : "false");
                edgeElement.setAttribute("id", "0");
                edgeElement = polygon.addElement("edge");
                edgeElement.setAttribute("p1", String.valueOf(v[1]));
                edgeElement.setAttribute(
                    "platform",
                    triangle.edges[1].getEdgeType() ? "true" : "false");
                edgeElement.setAttribute("id", "1");
                edgeElement = polygon.addElement("edge");
                edgeElement.setAttribute("p1", String.valueOf(v[0]));
                edgeElement.setAttribute(
                    "platform",
                    triangle.edges[2].getEdgeType() ? "true" : "false");
                edgeElement.setAttribute("id", "2");
            }

        }

        try
        {
            document.write(file);
            if (BlasterEditor.currentFrame == null)
                return;
            else
                JOptionPane.showInternalMessageDialog(
                    BlasterEditor.currentFrame.getContentPane(),
                    "Map written to file: " + file.getPath(),
                    "File saved",
                    JOptionPane.INFORMATION_MESSAGE);
        }
        catch (IOException e)
        {
            if (BlasterEditor.currentFrame == null)
                return;
            else
                JOptionPane.showInternalMessageDialog(
                    BlasterEditor.currentFrame.getContentPane(),
                    "Could not write file due to:\n" + e.getMessage(),
                    "Error",
                    JOptionPane.ERROR_MESSAGE);
        }

    }

}
```

Edge.java


```
/**
 * ------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------
 */

package dk.ruc.blaster.model;

import java.awt.Point;
import java.awt.Rectangle;
import java.util.HashMap;
/**
 * Represents an edge
 *
 * Last change by: $Author: dyregod $
 *
 * $Header: /var/cvs/Alwazah\040Editor/source/dk/ruc/blaster/model/Edge.java,v 1
.9 2002/12/16 00:19:08 dyregod Exp $
 *
 * @version $Revision: 1.9 $
 * @author dyregod
 */
public class Edge extends EditorObject
{
    Vertex a;
    Vertex b;
    private boolean isPlatform = false;

    /**
     * Constructor Edge.
     * @param vertex
     * @param vertex1
     */
    public Edge(Vertex a, Vertex b)
    {
        this.a = a;
        this.b = b;

    }

    /**
     * Returns the a vertex
     * @return Vertex
     */
    public Vertex getA()
    {
        return a;
    }

    public void setA( Vertex v )
    {
        a = v;
    }

    /**
     * Returns the b vertex
     * @return Vertex
     */
    public Vertex getB()
```

Edge.java Page 2/4

```
    {
        return b;
    }
    /**
     * Returns the normal to the edge
     * @return Vector
     */
    public Vector normal()
    {
        return b.position.subtract(a.position).normalize().ortho();
    }

    /**
     * @see dk.ruc.blaster.model.EditorObject#bounds()
     */
    public Rectangle bounds()
    {
        float ax = a.position.x;
        float ay = a.position.y;
        float bx = b.position.x;
        float by = b.position.y;

        if (ax > bx)
        {
            float temp = ax;
            ax = bx;
            bx = temp;
        }
        if (ay > by)
        {
            float temp = ay;
            ay = by;
            by = temp;
        }

        final int x = (int)(ax+0.5f);
        final int y = (int)(ay+0.5f);
        final int width = (int)((bx - ax)+0.5f);
        final int height = (int)((by - ay)+0.5f);

        return new Rectangle( x, y, width, height );
    }

    /**
     * Finds the distance from the edge to a point p
     * @param p
     * @return float
     */
    public float getDistanceToPoint( Point p )
    {
        if ( !bounds().contains( p ) )
        {
            return Float.MAX_VALUE;
        }

        final float x0 = (float) p.getX();
        final float y0 = (float) p.getY();

        final float x1 = a.position.x;
        final float y1 = a.position.y;
        final float x2 = b.position.x;
        final float y2 = b.position.y;
```

Edge.java Page 3/4

```
        // Calculate the distance between the point and the line.
        float dist = ( float ) (( Math.abs( (y2 - y1) * (x0 - x1) - (x2 - x1) *
(y0 - y1) ) )/ Math.sqrt( Math.pow( (x2 - x1), 2 ) + Math.pow( (y2 - y1), 2 ) ))
;

        return dist;
    }

    /**
     * @see dk.ruc.blaster.model.EditorObject#setProperties(java.util.HashMap)
     */
    public void setProperties(HashMap list)
    {
        String s = (String) list.get("platform");
        if ( s.equals("true") )
        {
            isPlatform = true;
        }
        else
        {
            isPlatform = false;
        }
    }

    /**
     * @see dk.ruc.blaster.model.EditorObject#getProperties()
     */
    public HashMap getProperties()
    {
        HashMap map = new HashMap();
        map.put("platform", isPlatform ? "true" : "false" );

        return map;
    }

    protected boolean selected = false;

    public void select( boolean value )
    {
        System.out.println("select " + value);
        selected = value;
    }

    /**
     * Returns <code>true</code> if edge is a platform
     * @return boolean
     */
    public boolean getEdgeType ()
    {
        return isPlatform;
    }

    public void move (int dx, int dy)
    {
        a.move( dx, dy );
        b.move( dx, dy );
    }

    public Rectangle updateOnDrag()
    {
        final Rectangle r = new Rectangle();

        r.add(a.updateOnDrag());
```

Edge.java


```
        r.add(b.updateOnDrag());

        return r;
    }
}
```

EditorObject.java


```
/**
 * ------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------
 */

package dk.ruc.blaster.model;

import java.awt.Rectangle;
import java.io.Serializable;
import java.util.HashMap;
/**
 * The super class of all objects in the editor
 *
 * Last change by: $Author: dyregod $
 *
 * $Header: /var/cvs/Alwazah\040Editor/source/dk/ruc/blaster/model/EditorObject.
java,v 1.5 2002/12/16 00:19:08 dyregod Exp $
 *
 * @version $Revision: 1.5 $
 * @author dyregod
 */
public abstract class EditorObject implements Serializable
{
    /**
     * Returns the objects bounding box
     * @return Rectangle
     */
    public abstract Rectangle bounds();
    /**
     * Sets this objects properties
     * @param list
     */
    public abstract void setProperties(HashMap list);
    /**
     * Get properties for this obejct
     * @return HashMap
     */
    public abstract HashMap getProperties();
    /**
     *
     */
    protected boolean selected = false;
    /**
     * Move object by dx, dy
     * @param dx
     * @param dy
     */
    public abstract void move (int dx, int dy);

    public abstract Rectangle updateOnDrag();

    public void select( boolean value )
    {
        System.out.println("select " + value);
        selected = value;
    }

}
```

Map.java


```
/**
 * ------------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------------
 */

package dk.ruc.blaster.model;

import java.awt.Dimension;
import java.awt.Point;
import java.awt.Rectangle;
import java.io.Serializable;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

import dk.ruc.blaster.ui.BlasterEditor;

public class Map extends EditorObject implements Serializable
{
    // list of stripes
    // list of fans
    // list of colors?
    // list of polygons
    public List vertices;
    public List triangles;

    public float gravityx = 0;
    public float gravityy = 0;
    public float width = 0;
    public float height = 0;
    public String name = "";

    private Dimension size;

    public Map()
    {
        width = 1000;
        height = 1000;
        vertices = new LinkedList();
        triangles = new LinkedList();
    }

    public Dimension getDimensions()
    {
        return new Dimension((int) width, (int) height);
    }
    public Rectangle bounds()
    {
        return new Rectangle((int) width, (int) height);
    }

    List getVerticesAtPoint(Point p)
    {
        List hit = new LinkedList();

        Vertex v;
```

Map.java


```
        Iterator i = vertices.iterator();

        while (i.hasNext())
        {
            v = (Vertex) i.next();

            if (v.contains(p))
            {
                hit.add(v);
            }
        }

        return hit;
    }

    public List getVerticesInRect(Rectangle bounds)
    {
        // At some point in the future, replace this with some sort of tree.
        // For now, just do a linear search and compare with bounds

        List visible = new LinkedList();

        Vertex v;

        Iterator i = vertices.iterator();

        while (i.hasNext())
        {
            v = (Vertex) i.next();

            if (bounds.intersects(v.bounds()))
            {
                visible.add(v);
            }
        }

        return visible;
    }

    Edge getNearestEdgeAtPoint(Point p)
    {
        float maxDist = Float.MAX_VALUE;

        Edge edge = null;

        Triangle t;

        Iterator i = triangles.iterator();

        while (i.hasNext())
        {
            t = (Triangle) i.next();

            for (int i2 = 0; i2 < t.edges.length; i2++)
            {
                float temp =
                    (float) Math.abs(t.edges[i2].getDistanceToPoint(p));
System.err.println("dist "+i2+" "+temp);
                if (temp < maxDist && temp <= 3.0f)
                {
                    maxDist = temp;
                    edge = t.edges[i2];
                }
```

```
            }
        }
        return edge;
    }

    List getTrianglesAtPoint(Point p)
    {
        LinkedList l = new LinkedList();

        Triangle t;

        Iterator i = triangles.iterator();

        while (i.hasNext())
        {
            t = (Triangle) i.next();

            if (t.contains(p))
            {
                l.add(t);
            }
        }

        return l;
    }

    public List getTrianglesInRect(Rectangle bounds)
    {
        // This needs to be improved

        LinkedList visible = new LinkedList();

        Triangle t;

        Iterator i = triangles.iterator();

        while (i.hasNext())
        {
            t = (Triangle) i.next();

            if (bounds.intersects(t.bounds()))
            {
                visible.add(t);
            }
        }

        return visible;
    }

    public EditorObject select(Point p)
    {
        EditorObject obj;
        obj = getNearestVertexToPoint(p, null);
        if (obj != null)
            return obj;

        obj = getNearestEdgeAtPoint(p);
        if (obj != null)
        {
            return obj;
        }

        List list = getTrianglesAtPoint(p);
```



```
        if (list.size() > 0)
        {
            obj = (EditorObject) list.get(0);
            return obj;
        }
        return this;
    }

    public Vertex getNearestVertexToPoint(Point p, Vertex ignore)
    {
        final Vertex temp = new Vertex(p.x, p.y);

        List vectors = getVerticesAtPoint(p);

        if (vectors.isEmpty())
        {
            return null;
        }

        Vertex v;
        Vertex nearestVertice = null;
        float nearestLength = Float.MAX_VALUE;
        float length;

        final Iterator i = vectors.iterator();

        while (i.hasNext())
        {
            v = (Vertex) i.next();

            length = temp.squaredDistanceTo(v);

            if (length < nearestLength && v != ignore)
            {
                nearestLength = length;
                nearestVertice = v;
            }
        }

        return nearestVertice;
    }

    public Vertex addVertex(Point p)
    {
        Vertex v = getNearestVertexToPoint(p, null);

        if (v == null)
        {
            v = new Vertex(p.x, p.y);

            // We didn't find any preexisting vertices, so it's ok to add the ne
w vertice to the map
            vertices.add(v);
        }

        return v;
    }

    public Triangle addTriangle(Vertex verts[])
    {
        final Triangle t = new Triangle(verts);

        triangles.add(t);
```

Map.java


```
        return t;
    }
    /**
     * @see dk.ruc.blaster.model.EditorObject#-(HashMap)
     */
    public void setProperties(HashMap list)
    {
        name = (String) list.get("name");
        gravityx = Float.valueOf((String) list.get("gravityx")).floatValue();
        gravityy = Float.valueOf((String) list.get("gravityy")).floatValue();
        width = Float.valueOf((String) list.get("width")).floatValue();
        height = Float.valueOf((String) list.get("height")).floatValue();
        if (BlasterEditor.currentFrame != null)
        {
            BlasterEditor.currentFrame.getCanvas().setSize(
                new Dimension((int) width, (int) height));
            BlasterEditor.currentFrame.setTitle(name);
        }
    }
    /**
     * @see dk.ruc.blaster.model.EditorObject#getProperties()
     */
    public HashMap getProperties()
    {
        HashMap map = new HashMap();
        map.put("width", String.valueOf(width));
        map.put("height", String.valueOf(height));
        map.put("gravityx", String.valueOf(gravityx));
        map.put("gravityy", String.valueOf(gravityy));
        map.put("name", name);
        return map;
    }
    public void unselectall()
    {
        for (Iterator iter = vertices.iterator(); iter.hasNext();)
        {
            EditorObject element = (EditorObject) iter.next();
            element.select(false);
        }
        for (Iterator iter = triangles.iterator(); iter.hasNext();)
        {
            EditorObject element = (EditorObject) iter.next();
            element.select(false);
        }
    }

    public void move(int dx, int dy)
    {
    }

    public Rectangle updateOnDrag()
    {
        return null;
    }
}
```

Triangle.java


```
/**
 * ----------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ----------------------------------------------------------------------------
 */

package dk.ruc.blaster.model;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.Polygon;
import java.awt.Rectangle;
import java.io.Serializable;
import java.util.HashMap;

public class Triangle extends EditorObject implements Serializable
{
    private final Color selectedColor = new Color( 0.8f, 1.0f, 0.8f );
    private final Color fillColor = Color.white;
    private final Color borderColor = Color.black;

    private Color color = Color.white;

    public Vertex vertices[];
    public Edge edges[];
    private Polygon poly = null;
    private Rectangle bounds = null;


    public Triangle( Vertex verts[] )
    {
        vertices = verts;
        edges = new Edge[ verts.length ];

        for ( int i = 0; i < verts.length; i++ )
        {
            verts[i].addPoly( this );
            edges[i] = new Edge( verts[i], verts[(i+1)%verts.length] );
        }

        update();
    }

    public void update()
    {
        // reorder verts to maintain clockwise orientation?

        final int xcoords[] = new int[4];
        final int ycoords[] = new int[4];

        for ( int i = 0; i < 3; i++ )
        {
            xcoords[i] = (int)vertices[i].position.x;
            ycoords[i] = (int)vertices[i].position.y;
        }

        xcoords[3] = xcoords[0];
        ycoords[3] = ycoords[0];
```

```
        poly = new Polygon( xcoords, ycoords, 4 );
        bounds = poly.getBounds();
    }

    public boolean contains( Point p )
    {
        return poly.contains( p );
    }

    public Rectangle bounds()
    {
        return bounds;
    }

    public void draw( Graphics g )
    {
        final Color c = g.getColor();

        g.setColor( selected ? selectedColor : color );
        g.fillPolygon( poly );

        g.setColor( borderColor );
        g.drawPolygon( poly );

        g.setColor( c );
    }
    /**
     * @see dk.ruc.blaster.model.EditorObject#setProperties(HashMap)
     */
    public void setProperties(HashMap list)
    {
        float r = Float.valueOf((String)list.get("r")).floatValue();
        System.out.println(r);
        float g = Float.valueOf((String)list.get("g")).floatValue();
        float b = Float.valueOf((String)list.get("b")).floatValue();

        setColor(new Color(r,g,b));
    }

    /**
     * Returns the color.
     * @return Color
     */
    public Color getColor()
    {
        return color;
    }

    /**
     * Sets the color.
     * @param color The color to set
     */
    public void setColor(Color color)
    {
        this.color = color;
    }

    /**
     * @see dk.ruc.blaster.model.EditorObject#getProperties()
     */
    public HashMap getProperties()
    {
```



```
        HashMap map = new HashMap();
        float[] comp = new float[3];
        color.getRGBColorComponents(comp);
        map.put("r", String.valueOf(comp[0]));
        map.put("g", String.valueOf(comp[1]));
        map.put("b", String.valueOf(comp[2]));
        return map;
    }

    public void move (int dx, int dy)
    {
        for ( int i = 0; i < vertices.length; i++ )
        {
            vertices[i].move( dx, dy );
        }
    }

    public Rectangle updateOnDrag()
    {
        final Rectangle r = new Rectangle();

        for( int i = 0; i < vertices.length; i++ )
        {
            r.add( vertices[i].updateOnDrag() );
        }

        return r;
    }

}
```

Vector.java


```
/**
 * ------------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------------
 */

package dk.ruc.blaster.model;

import java.io.Serializable;

public class Vector implements Serializable
{
    public float x;
    public float y;

    public Vector( float X, float Y )
    {
        x = X;
        y = Y;
    }

    public Vector add( Vector v )
    {
        return new Vector( this.x + v.x, this.y + v.y );
    }

    public Vector subtract( Vector v )
    {
        return new Vector( this.x - v.x, this.y - v.y );
    }

    public float length2()
    {
        return x*x + y*y;
    }

    public float length()
    {
        return (float)Math.sqrt( length2() );
    }
    public Vector scale(float scale)
    {
        return new Vector(x*scale, y*scale);
    }
    public Vector normalize()
    {
        return this.scale(1/length());
    }
    public Vector ortho()
    {
        return new Vector(-y,x);
    }
    public float dot(Vector b)
    {
        return x*b.x + y*b.y;
    }

}
```

Vertex.java


```
/**
 * ------------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------------
 */

package dk.ruc.blaster.model;

import java.awt.Graphics;
import java.awt.Point;
import java.awt.Rectangle;
import java.io.Serializable;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

public class Vertex extends EditorObject implements Serializable
{
    Vector position;

    private boolean selected = false;

    private List triangles = new LinkedList();

    final public static int THRESHHOLD = 3;


    public Vertex( float a, float b )
    {
        position = new Vector( a, b );
    }

    public void select( boolean value )
    {
        selected = value;
    }

    public void move( int dx, int dy )
    {
        position.x += dx;
        position.y += dy;
    }

    public void draw( Graphics g )
    {
        final int ix = (int)(position.x+0.5f);
        final int iy = (int)(position.y+0.5f);

        g.drawRect( ix-1, iy-1, 2, 2 );
    }

    public void addPoly( Triangle t )
    {
        triangles.add( t );
    }

    public void removePoly(Triangle t)
    {
```

```
        triangles.remove(t);
    }

    public Rectangle updateOnDrag()
    {
        Triangle t;

        final Rectangle r = new Rectangle();

        Iterator i = triangles.iterator();

        while ( i.hasNext() )
        {
            t = (Triangle)i.next();

            r.add( t.bounds() );
            t.update();
        }

        return r;
    }

    public Rectangle bounds()
    {
        final int ix = (int)(position.x+0.5f);
        final int iy = (int)(position.y+0.5f);

        return new Rectangle( ix-THRESHHOLD, iy-THRESHHOLD, THRESHHOLD*2, THRESH
HOLD*2 );
    }

    public boolean contains( Point p )
    {
        return bounds().contains( p );
    }

    public float squaredDistanceTo( Vertex v )
    {
        final Vector vec = position.subtract( v.position );

        return vec.length2();
    }
    /**
     * @see dk.ruc.blaster.model.EditorObject#setProperties(HashMap)
     */
    public void setProperties(HashMap list)
    {
    }

    /**
     * @see dk.ruc.blaster.model.EditorObject#getProperties()
     */
    public HashMap getProperties()
    {
        return null;
    }

    /**
    * Returns the position.
    * @return Vector
    */
    public Vector getPosition()
    {
```



```
        return position;
    }

    public int getNumberOfAttachedPolygons()
    {
        return triangles.size();
    }
}
```

AbstractMode.java


```
/**
 * ------------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------------
 */

package dk.ruc.blaster.ui;

import java.awt.Rectangle;
import java.awt.event.KeyEvent;
import java.awt.event.MouseEvent;

import dk.ruc.blaster.model.*;

abstract public class AbstractMode
{
    protected Map map;

    public AbstractMode( Map m )
    {
        map = m;
    }

    public void activate()
    {
    }

    public void deactivate()
    {
    }

    public Rectangle keyPressed( KeyEvent e )
    {
        return null;
    }

    public Rectangle keyReleased( KeyEvent e )
    {
        return null;
    }

    public Rectangle keyTyped( KeyEvent e )
    {
        return null;
    }

    public Rectangle mousePressed( MouseEvent e )
    {
        return null;
    }

    public Rectangle mouseReleased( MouseEvent e )
    {
        return null;
    }

    public Rectangle mouseClicked( MouseEvent e )
    {
        return null;
```

AbstractMode.java


```
    }

    public Rectangle mouseDragged( MouseEvent e )
    {
        return null;
    }

    public Rectangle mouseMoved( MouseEvent e )
    {
        return null;
    }

    public Rectangle mouseEntered( MouseEvent e )
    {
        return null;
    }

    public Rectangle mouseExited( Rectangle e )
    {
        return null;
    }
}
```

```
/**
 * ------------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------------
 */

package dk.ruc.blaster.ui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyVetoException;
import java.beans.VetoableChangeListener;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JDesktopPane;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JToolBar;

import dk.ruc.blaster.export.Exporter;
import dk.ruc.blaster.model.Map;
import dk.ruc.blaster.ui.property.PropertyFloat;

public class BlasterEditor
    extends JFrame
    implements VetoableChangeListener, ComponentListener
{
    JDesktopPane desktop = new JDesktopPane();
    MapFrame mapFrame;

    public static MapFrame currentFrame;
    public static int windowCount = 0;

    static void switchMode()
    {
        if (currentFrame == null)
            return;
        currentFrame.switchMode();

    }
```



```
    public void buildUI()
    {
        this.setTitle("BlasterEditor - The One and Only");

        JPanel panel1 = new JPanel();
        panel1.setLayout(new BorderLayout());
        JPanel panel2 = new JPanel();
        panel2.setLayout(new BorderLayout());
        panel1.add(createMenuBar(), BorderLayout.NORTH);
        panel2.add(createToolBar(), BorderLayout.CENTER);
        panel1.add(panel2, BorderLayout.SOUTH);

        desktop.add(FloatingToolBar.getInstance());
        desktop.add(PropertyFloat.getInstance());

        desktop.setBorder(BorderFactory.createEtchedBorder());
        FloatingToolBar.getInstance().setSize(60, 60);
        FloatingToolBar.getInstance().setVisible(true);
        FloatingToolBar.getInstance().setLocation(20, 60);

        this.getContentPane().add(panel1, BorderLayout.NORTH);
        this.getContentPane().add(desktop, BorderLayout.CENTER);

        this.setSize(640, 480);
        Dimension size = Toolkit.getDefaultToolkit().getScreenSize();
        this.setLocation(
            (size.width - this.getSize().width) / 2,
            (size.height - this.getSize().height) / 2);
    }

    public JMenuBar createMenuBar()
    {
        JMenuBar menubar = new JMenuBar();
        JMenu filemenu = new JMenu();
        filemenu.setText("File");
        JMenuItem itemNew = new JMenuItem();
        itemNew.setText("New");
        itemNew.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                newFrame();
            }
        });
        JMenuItem itemOpen = new JMenuItem();
        itemOpen.setText("Open");
        itemOpen.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent ae)
            {
                load();
            }
        });

        JMenuItem itemSave = new JMenuItem();
        itemSave.setText("Save");
        itemSave.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                save();
            }
```

```
        });

        JMenuItem itemExport = new JMenuItem();
        itemExport.setText("Export");
        itemExport.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                export();
            }
        });

        JMenuItem itemExit = new JMenuItem();
        itemExit.setText("Exit");

        filemenu.add(itemNew);
        filemenu.addSeparator();
        filemenu.add(itemOpen);
        filemenu.add(itemSave);
        filemenu.addSeparator();
        filemenu.add(itemExport);
        filemenu.addSeparator();
        filemenu.add(itemExit);

        menubar.add(filemenu);

        return menubar;
    }
    public JToolBar createToolBar()
    {
        JToolBar toolbar = new JToolBar();
        JButton buttonNew = new JButton("New");
        JButton buttonOpen = new JButton("Open");
        JButton buttonSave = new JButton("Save");
        JButton buttonExport = new JButton("Export");
        buttonNew.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                newFrame();
            }
        });
        buttonOpen.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                load();
            }
        });
        buttonSave.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                save();
            }
        });
        buttonExport.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                export();
            }
        });
```



```
        toolbar.add(buttonNew);
        toolbar.add(buttonOpen);
        toolbar.add(buttonSave);
        toolbar.add(buttonExport);

        return toolbar;
    }
    private void load()
    {
        try
        {
            boolean test = false;
            if (currentFrame != null)
            {
                test = currentFrame.getLastUsedDir() == null;
            }
            else
                test = true;
            JFileChooser filechooser =
                new JFileChooser(
                    test
                        ? new File(System.getProperty("user.dir"))
                        : currentFrame.getLastUsedDir());
            filechooser.setFileFilter(
                new FileFilter(".bin", "Editor binaries"));
            filechooser.showOpenDialog(mapFrame);
            File file = filechooser.getSelectedFile();
            if (file == null)
                return;
            FileInputStream istream = new FileInputStream(file);
            ObjectInputStream p = new ObjectInputStream(istream);

            Map map = (Map) p.readObject();

            istream.close();

            MapFrame frame = new MapFrame(map);
            desktop.add(frame, new Integer(-1));
            frame.setSize(400, 300);
            frame.display();
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }
    }
    private void export()
    {
        boolean test = false;
        if (currentFrame != null)
        {
            test = currentFrame.getLastUsedDir() == null;
        }
        else
            test = true;
```

```
        JFileChooser filechooser =
            new JFileChooser(
                test
                    ? new File(System.getProperty("user.dir"))
                    : currentFrame.getLastUsedDir());
        filechooser.setFileFilter(new FileFilter(".map", "Blaster Map files"));
        filechooser.showSaveDialog(mapFrame);
        File file = filechooser.getSelectedFile();
        if (file == null)
            return;
        currentFrame.setLastUsedDir(file.getParentFile());
        Exporter.export(currentFrame.map, file);
    }
    private void save()
    {
        try
        {

            boolean test = false;
            if (currentFrame != null)
            {
                test = currentFrame.getLastUsedDir() == null;
            }
            else
                test = true;
            JFileChooser filechooser =
                new JFileChooser(
                    test
                        ? new File(System.getProperty("user.dir"))
                        : currentFrame.getLastUsedDir());
            filechooser.setFileFilter(
                new FileFilter(".bin", "Editor binaries"));
            filechooser.showSaveDialog(mapFrame);
            File file = filechooser.getSelectedFile();
            if (file == null)
                return;
            currentFrame.setLastUsedDir(file.getParentFile());
            FileOutputStream ostream = new FileOutputStream(file);
            ObjectOutputStream p = new ObjectOutputStream(ostream);

            p.writeObject(currentFrame.map);

            p.flush();
            ostream.close();
            JOptionPane.showInternalMessageDialog(
                BlasterEditor.currentFrame.getContentPane(),
                "Map written to file: " + file.getPath(),
                "File saved",
                JOptionPane.INFORMATION_MESSAGE);
        }
        catch (Exception ee)
        {
            JOptionPane.showInternalMessageDialog(
                BlasterEditor.currentFrame.getContentPane(),
                "Could not write file due to:\n" + ee.getMessage(),
                "Error",
                JOptionPane.ERROR_MESSAGE);
        }
    }
    private void newFrame()
    {
        MapFrame frame = new MapFrame("Untitled " + windowCount);
        desktop.add(frame, new Integer(-1));
```



```
        frame.setSize(400, 300);
        frame.display();
    }
    public static void main(String[] args)
    {
        BlasterEditor editor = new BlasterEditor();
        /*      editor.addWindowListener(new WindowAdapter()
                {
                    public void windowClosing(WindowEvent e)
                    {
                        System.exit(0);
                    }
                });*/
        editor.buildUI();
        editor.setVisible(true);
    }
    /**
     * @see java.awt.event.ComponentListener#componentHidden(ComponentEvent)
     */
    public void componentHidden(ComponentEvent e)
    {
    }

    /**
     * @see java.awt.event.ComponentListener#componentMoved(ComponentEvent)
     */
    public void componentMoved(ComponentEvent e)
    {
    }

    /**
     * @see java.awt.event.ComponentListener#componentResized(ComponentEvent)
     */
    public void componentResized(ComponentEvent e)
    {
    }

    /**
     * @see java.awt.event.ComponentListener#componentShown(ComponentEvent)
     */
    public void componentShown(ComponentEvent e)
    {
    }

    /**
     * @see java.beans.VetoableChangeListener#vetoableChange(PropertyChangeEvent
)
     */
    public void vetoableChange(PropertyChangeEvent evt)
        throws PropertyVetoException
    {

    }

}
class FileFilter extends javax.swing.filechooser.FileFilter
{
    private String suffix;
    private String description;

    public FileFilter(String suffix, String description)
    {
        this.suffix = suffix;
```

BlasterEditor.java


```
        this.description = description;
    }
    public boolean accept(File f)
    {
        if (f.isDirectory())
            return true;
        else
            return f.getPath().endsWith(suffix);
    }

    public String getDescription()
    {
        return description;
    }
}
```

EditorCanvas.java


```
/**
 * ------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------
 */

package dk.ruc.blaster.ui;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.util.Iterator;
import java.util.List;

import javax.swing.JPanel;

import dk.ruc.blaster.model.Map;
import dk.ruc.blaster.model.Triangle;
import dk.ruc.blaster.model.Vertex;

public class EditorCanvas extends JPanel
{
    final private Map map;

    public EditorCanvas(Map map)
    {
        this.map = map;
    }

    public Dimension getPreferredSize()
    {
        return map.getDimensions();
    }

    public Dimension getMinimumSize()
    {
        return map.getDimensions();
    }

    public Dimension getMaximumSize()
    {
        return map.getDimensions();
    }

    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);

        // Get the part of the canvas that needs to be redrawn
        Rectangle bounds = g.getClipBounds();

        // Draw the axes
        Dimension d = map.getDimensions();
        g.drawLine( d.width/2, 0, d.width/2, d.height );
        g.drawLine( 0, d.height/2, d.width, d.height/2 );

        // Repaint the polygons that lie within the bounds
        List verts = map.getVerticesInRect(bounds);
        List tris = map.getTrianglesInRect(bounds);
```

EditorCanvas.java


```
        Iterator i = null;

        Triangle t = null;

        i = tris.iterator();

        while (i.hasNext())
        {
            t = (Triangle) i.next();

            t.draw(g);
        }

        Vertex v = null;

        i = verts.iterator();

        while (i.hasNext())
        {
            v = (Vertex) i.next();

            v.draw(g);
        }
    }
}
```

FloatingToolBar.java


```
/**
 * ------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------
 */

package dk.ruc.blaster.ui;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.beans.PropertyVetoException;

import javax.swing.ButtonGroup;
import javax.swing.JInternalFrame;
import javax.swing.JToggleButton;

/*
 * Floating toolbar class.
 *
 * Last change by: $Author$
 *
 * $Header$
 *
 * @version $Revision$
 * @author dk13043
 */
public class FloatingToolBar extends JInternalFrame
{
    private static FloatingToolBar instance = null;

    private JToggleButton polygonModeButton;
    private JToggleButton moveModeButton;

    public static FloatingToolBar getInstance()
    {
        if (instance == null)
        {
            instance = new FloatingToolBar();
        }
        return instance;
    }
    private FloatingToolBar()
    {
        getContentPane().setLayout(new GridLayout(2, 2));

        ButtonGroup group = new ButtonGroup();

        polygonModeButton = new JToggleButton("P");
        polygonModeButton.addActionListener(new ActionListener()
        {

            public void actionPerformed(ActionEvent e)
            {
                BlasterEditor.currentFrame.getCurrentMode().deactivate();
                BlasterEditor.currentFrame.setCurrentMode(BlasterEditor.currentF
rame.polygonMode);
                BlasterEditor.currentFrame.getCurrentMode().activate();
                BlasterEditor.currentFrame.grabFocus();
```

```
                BlasterEditor.currentFrame.moveToFront();
                try
                {
                    BlasterEditor.currentFrame.setSelected(true);
                }
                catch (PropertyVetoException ee)
                {
                }
            }
        });
        polygonModeButton.setSelected(true);
        group.add(polygonModeButton);
        getContentPane().add(polygonModeButton);
        moveModeButton = new JToggleButton("M");
        moveModeButton.addActionListener(new ActionListener()
        {

            public void actionPerformed(ActionEvent e)
            {
                BlasterEditor.currentFrame.getCurrentMode().deactivate();
                BlasterEditor.currentFrame.setCurrentMode(BlasterEditor.currentF
rame.moveMode);
                BlasterEditor.currentFrame.getCurrentMode().activate();
                BlasterEditor.currentFrame.grabFocus();
                BlasterEditor.currentFrame.moveToFront();
                try
                {
                    BlasterEditor.currentFrame.setSelected(true);
                }
                catch (PropertyVetoException ee)
                {
                }
            }
        });
        group.add(moveModeButton);
        getContentPane().add(moveModeButton);

        this.putClientProperty("JInternalFrame.isPalette", Boolean.TRUE);

    }
    public void refresh()
    {
        if (BlasterEditor.currentFrame.getCurrentMode() instanceof PolygonMode)
            polygonModeButton.setSelected(true);
        if (BlasterEditor.currentFrame.getCurrentMode() instanceof MoveMode)
            moveModeButton.setSelected(true);
    }
}
```



```
/**
 * ------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------
 */

package dk.ruc.blaster.ui;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.MouseEvent;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyVetoException;
import java.beans.VetoableChangeListener;
import java.io.File;

import javax.swing.JInternalFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JViewport;
import javax.swing.event.InternalFrameEvent;
import javax.swing.event.InternalFrameListener;
import javax.swing.event.MouseInputAdapter;

import dk.ruc.blaster.model.EditorObject;
import dk.ruc.blaster.model.Map;

/*
 * dk13043 did not update javadoc, and $Author$ didn't care :)
 *
 * Last change by: $Author$
 *
 * $Header$
 *
 * @version $Revision$
 * @author dk13043
 */
public class MapFrame
    extends JInternalFrame
    implements InternalFrameListener, VetoableChangeListener
{
    Map map;
    private AbstractMode currentMode;
    final public PolygonMode polygonMode;
    final public MoveMode moveMode;
    private String title;
    private JScrollPane scrollView;
    private EditorCanvas canvas;
    private File lastUsedDir = null;

    private EditorObject currentSelection = null;
    /**
     * Constructor for MapFrame.
     */
    public MapFrame(String title)
```

```
    {
        this.title = title;
        map = new Map();
        moveMode = new MoveMode(map);
        polygonMode = new PolygonMode(map);
        init();
    }
    public MapFrame(Map map)
    {
        this.map = map;
        this.title = map.name;
        moveMode = new MoveMode(map);
        polygonMode = new PolygonMode(map);
        init();
    }

    private void init()
    {
        addVetoableChangeListener(this);
        currentMode = polygonMode;
        addInternalFrameListener(this);

        canvas = new EditorCanvas(map);
        canvas.setSize(1024, 768);
        scrollView = new JScrollPane(canvas);
        scrollView.setHorizontalScrollBarPolicy(
            JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        scrollView.setVerticalScrollBarPolicy(
            JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        scrollView.setMinimumSize(new Dimension(100, 100));
        scrollView.setPreferredSize(new Dimension(320, 240));

        getContentPane().add(scrollView, BorderLayout.CENTER);

        this.setClosable(true);
        this.setMaximizable(true);
        this.setIconifiable(true);
        this.setResizable(true);

        KeyAdapter keyHandler = new KeyAdapter()
        {
            public void keyPressed(KeyEvent e)
            {
                final Rectangle bounds = currentMode.keyPressed(e);

                if (bounds != null)
                    canvas.repaint(
                        bounds.x,
                        bounds.y,
                        bounds.width + 1,
                        bounds.height + 1);
            }

            public void keyReleased(KeyEvent e)
            {
                final Rectangle bounds = currentMode.keyReleased(e);

                if (bounds != null)
                    canvas.repaint(
                        bounds.x,
                        bounds.y,
                        bounds.width + 1,
                        bounds.height + 1);
```



```
            }

            public void keyTyped(KeyEvent e)
            {
                final Rectangle bounds = currentMode.keyTyped(e);

                if (bounds != null)
                    canvas.repaint(
                        bounds.x,
                        bounds.y,
                        bounds.width + 1,
                        bounds.height + 1);
            }
        };

        MouseInputAdapter inputHandler = new MouseInputAdapter()
        {
            public void mousePressed(MouseEvent e)
            {
                final Rectangle bounds = currentMode.mousePressed(e);
                if (bounds != null)
                    canvas.repaint(
                        bounds.x - 1,
                        bounds.y - 1,
                        bounds.width + 2,
                        bounds.height + 2);
            }
            public void mouseReleased(MouseEvent e)
            {
                final Rectangle bounds = currentMode.mouseReleased(e);
                if (bounds != null)
                    canvas.repaint(
                        bounds.x - 1,
                        bounds.y - 1,
                        bounds.width + 2,
                        bounds.height + 2);
            }
            public void mouseClicked(MouseEvent e)
            {
                final Rectangle bounds = currentMode.mouseClicked(e);
                if (bounds != null)
                    canvas.repaint(
                        bounds.x - 1,
                        bounds.y - 1,
                        bounds.width + 2,
                        bounds.height + 2);
            }
            public void mouseDragged(MouseEvent e)
            {
                final Rectangle bounds = currentMode.mouseDragged(e);
                if (bounds != null)
                    canvas.repaint(
                        bounds.x - 1,
                        bounds.y - 1,
                        bounds.width + 2,
                        bounds.height + 2);
            }
            public void mouseMoved(MouseEvent e)
            {
                final Rectangle bounds = currentMode.mouseMoved(e);
                if (bounds != null)
                    canvas.repaint(
                        bounds.x - 1,
```

```
                        bounds.y - 1,
                        bounds.width + 2,
                        bounds.height + 2);
            }
            public void mouseEntered(MouseEvent e)
            {
                final Rectangle bounds = currentMode.mouseEntered(e);
                if (bounds != null)
                    canvas.repaint(
                        bounds.x - 1,
                        bounds.y - 1,
                        bounds.width + 2,
                        bounds.height + 2);
            }
            public void mouseExited(MouseEvent e)
            {
                final Rectangle bounds = currentMode.mouseEntered(e);
                if (bounds != null)
                    canvas.repaint(
                        bounds.x - 1,
                        bounds.y - 1,
                        bounds.width + 2,
                        bounds.height + 2);
            }
        };
        canvas.addMouseListener(inputHandler);
        canvas.addMouseMotionListener(inputHandler);
        canvas.addKeyListener(keyHandler);

    }
    public void display()
    {
        try
        {
            super.setVisible(true);
            this.setMaximum(true);

            final JViewport v = scrollView.getViewport();
            final Dimension canvasSize = canvas.getSize();
            final Rectangle view = v.getViewRect();

            final int vx = (canvasSize.width - view.width) / 2;
            final int vy = (canvasSize.height - view.height) / 2;

            view.setLocation(new Point(vx, vy));

            v.scrollRectToVisible(view);
            canvas.requestFocus();
        }
        catch (Exception e)
        {
        }
    }
    public void switchMode()
    {
        if (currentMode instanceof PolygonMode)
        {
            currentMode.deactivate();
            currentMode = moveMode;
            currentMode.activate();
        }
        else
        {
```



```
            currentMode.deactivate();
            currentMode = polygonMode;
            currentMode.activate();
        }
    }
    /**
     * Returns the currentMode.
     * @return AbstractMode
     */
    public AbstractMode getCurrentMode()
    {
        return currentMode;
    }

    /**
     * Sets the currentMode.
     * @param currentMode The currentMode to set
     */
    public void setCurrentMode(AbstractMode currentMode)
    {
        this.currentMode = currentMode;
    }

    /**
     * @see javax.swing.event.InternalFrameListener#internalFrameActivated(InternalFrameEvent)
     */
    public void internalFrameActivated(InternalFrameEvent e)
    {
        BlasterEditor.currentFrame = this;
        FloatingToolBar.getInstance().refresh();
    }

    /**
     * @see javax.swing.event.InternalFrameListener#internalFrameClosed(InternalFrameEvent)
     */
    public void internalFrameClosed(InternalFrameEvent e)
    {

    }

    /**
     * @see javax.swing.event.InternalFrameListener#internalFrameClosing(InternalFrameEvent)
     */
    public void internalFrameClosing(InternalFrameEvent e)
    {
    }

    /**
     * @see javax.swing.event.InternalFrameListener#internalFrameDeactivated(InternalFrameEvent)
     */
    public void internalFrameDeactivated(InternalFrameEvent e)
    {
    }

    /**
     * @see javax.swing.event.InternalFrameListener#internalFrameDeiconified(InternalFrameEvent)
     */
    public void internalFrameDeiconified(InternalFrameEvent e)
```

```
    {
        BlasterEditor.currentFrame = this;
    }

    /**
     * @see javax.swing.event.InternalFrameListener#internalFrameIconified(InternalFrameEvent)
     */
    public void internalFrameIconified(InternalFrameEvent e)
    {
    }

    /**
     * @see javax.swing.event.InternalFrameListener#internalFrameOpened(InternalFrameEvent)
     */
    public void internalFrameOpened(InternalFrameEvent e)
    {
        BlasterEditor.windowCount++;
        BlasterEditor.currentFrame = this;
    }

    public void vetoableChange(PropertyChangeEvent pce)
        throws PropertyVetoException
    {

        if (pce.getPropertyName().equals(IS_CLOSED_PROPERTY))
        {

            int option =
                JOptionPane.showInternalConfirmDialog(
                    this,
                    "Do you really want to close?",
                    "Warning",
                    JOptionPane.YES_NO_OPTION,
                    JOptionPane.WARNING_MESSAGE);

            if (option != JOptionPane.YES_OPTION)
            {
                throw new PropertyVetoException("User cancelled", pce);
            }

        }
    }
    /**
     * Returns the title.
     * @return String
     */
    public String getTitle()
    {
        return title;
    }

    /**
     * Sets the title.
     * @param title The title to set
     */
    public void setTitle(String title)
    {
        this.title = title;
    }

    /**
```



```
     * Returns the currentSelection.
     * @return EditorObject
     */
    public EditorObject getCurrentSelection()
    {
        return currentSelection;
    }

    /**
     * Sets the currentSelection.
     * @param currentSelection The currentSelection to set
     */
    public void setCurrentSelection(EditorObject currentSelection)
    {
        this.currentSelection = currentSelection;
    }

    /**
     * Returns the lastUsedDir.
     * @return File
     */
    public File getLastUsedDir()
    {
        return lastUsedDir;
    }

    /**
     * Sets the lastUsedDir.
     * @param lastUsedDir The lastUsedDir to set
     */
    public void setLastUsedDir(File lastUsedDir)
    {
        this.lastUsedDir = lastUsedDir;
    }

    /**
     * Returns the canvas.
     * @return EditorCanvas
     */
    public EditorCanvas getCanvas()
    {
        return canvas;
    }
}
```

MoveMode.java


```
/**
 * ------------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------------
 */

package dk.ruc.blaster.ui;

import java.awt.Point;
import java.awt.Rectangle;
import java.awt.event.MouseEvent;
import java.awt.event.KeyEvent;
import java.util.Iterator;

import dk.ruc.blaster.model.EditorObject;
import dk.ruc.blaster.model.Map;
import dk.ruc.blaster.model.Triangle;
import dk.ruc.blaster.model.Vertex;
import dk.ruc.blaster.ui.property.PropertyFloat;

public class MoveMode extends AbstractMode
{
    private EditorObject selectedObject = null;

    private Point lastPosition;
    private Point startPosition;

    public MoveMode(Map m)
    {
        super(m);
    }

    public void activate()
    {
        if (selectedObject != null)
        {
            selectedObject.select(false);
        }

        System.out.println("MoveMode activated");
    }

    public void deactivate()
    {
        if (selectedObject != null)
        {
            selectedObject.select(false);
        }

        System.out.println("MoveMode deactivated");
    }

    public Rectangle mouseDragged(MouseEvent e)
    {

        final Point currentPosition = e.getPoint();
        final Rectangle r = new Rectangle();
        final int dx = currentPosition.x - lastPosition.x;
        final int dy = currentPosition.y - lastPosition.y;
```

MoveMode.java


```
        if (selectedObject == null)
        {
            // drag box?
        }
        else
        {
            selectedObject.move(dx, dy);
            Rectangle rr = selectedObject.updateOnDrag();
            if(rr != null)
                r.add(rr);
        }
        r.add(lastPosition);
        r.add(currentPosition);

        lastPosition = currentPosition;
        return r;
    }

    public Rectangle mousePressed(MouseEvent e)
    {

        Point q = e.getPoint();
        EditorObject lastSelected = selectedObject;
        map.unselectall();
        EditorObject v = map.select(q);

        selectedObject = v;

        lastPosition = startPosition = q;

        if (v == null)
        {
            return null;
        }

        v.select(true);
        PropertyFloat.getInstance().setData(v.getProperties());
        BlasterEditor.currentFrame.setCurrentSelection(v);
        /*System.out.println(
            "Mouse pressed, "
                + (selectedObject == null ? "0" : "1")
                + " object(s) selected");*/

        if (lastSelected != null)
        {
            Rectangle r = v.bounds();
            r.add(lastSelected.bounds());
            return r;
        }
        else
            return v.bounds();

    }

    public Rectangle mouseReleased( MouseEvent e )
    {
        if (selectedObject instanceof Vertex)
        {
            Vertex v = (Vertex) selectedObject;
            final Point currentPosition = e.getPoint();
            Vertex nearestVertex = map.getNearestVertexToPoint(currentPosition,
v);
```

```
            if (nearestVertex == null)
            {
                return null;
            }

            if ( v.bounds().intersects( nearestVertex.bounds() ) )
            {
                boolean update = false;

                Triangle t;
                Iterator i = map.triangles.iterator();

                while (i.hasNext())
                {
                    t = (Triangle) i.next();

                    for( int i2 = 0; i2 < t.edges.length; i2++)
                    {
                        if (v == t.edges[i2].getA())
                        {
                            t.edges[i2].setA(nearestVertex);
                            update = true;
                        }
                    }

                    for( int i2 = 0; i2 < t.vertices.length; i2++)
                    {
                        if (v == t.vertices[i2])
                        {
                            t.vertices[i2] = nearestVertex;
                            update = true;
                        }
                    }

                    if ( update )
                    {
                        nearestVertex.addPoly(t);
                        nearestVertex.updateOnDrag();
                    }
                }
            map.vertices.remove(v);
            }
        }
    return null;
    }

    /**
     * @see dk.ruc.blaster.ui.AbstractMode#keyPressed(KeyEvent)
     */
    public Rectangle keyPressed(KeyEvent e)
    {
    if (e.getKeyCode() == KeyEvent.VK_DELETE)
        {
            if (selectedObject instanceof Triangle)
            {
                Triangle t = (Triangle) selectedObject;
                    for (int i2 = 0; i2 < t.vertices.length; i2++)
                    {
                        t.vertices[i2].removePoly( t );

                        if (t.vertices[i2].getNumberOfAttachedPolygons() == 0)
                        {
                            map.vertices.remove(t.vertices[i2]);
```



```
                        }
                    }
                }
                map.triangles.remove(selectedObject);

            }
            return selectedObject.bounds();
        }

        /**
         * @see dk.ruc.blaster.ui.AbstractMode#keyReleased(KeyEvent)
         */
        public Rectangle keyReleased(KeyEvent e) {
            System.out.println("released: "+e.getKeyCode());
            return super.keyReleased(e);
        }

}
```

PolygonMode.java


```
/**
 * ------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------
 */

package dk.ruc.blaster.ui;

import java.awt.Rectangle;
import java.awt.event.MouseEvent;

import dk.ruc.blaster.model.Map;
import dk.ruc.blaster.model.Triangle;
import dk.ruc.blaster.model.Vertex;

public class PolygonMode extends AbstractMode
{
    private Vertex verts[];
    private int vertIndex;

    public PolygonMode( Map m )
    {
        super( m );
        reset();
    }

    private void reset()
    {
        verts = new Vertex[]{ null, null, null };
        vertIndex = 0;
    }

    public void activate()
    {
        reset();
    }

    public void deactivate()
    {
        reset();
    }

    public Rectangle mouseClicked( MouseEvent e )
    {
        System.err.println("clicked "+vertIndex);
        // Place a vertex where the user clicked. If the user clicked on an pree
xisting vertex,
        // it will be returned.
        final Vertex v = map.addVertex( e.getPoint() );

        // The part of the window that needs updating when we're done
        final Rectangle bounds = v.bounds();

        // Add the current vertex to our list of vertices
        verts[vertIndex++] = v;

        // When we reach three vertices, create a polygon and reset the list
        if ( vertIndex == 3 )
        {
```

PolygonMode.java


```
            final Triangle t = map.addTriangle( verts );

            bounds.add( t.bounds() );

            reset();
        }

        return bounds;
    }
}
```

```
/**
 * ------------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------------
 */

package dk.ruc.blaster.ui.property;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

import javax.swing.DefaultCellEditor;
import javax.swing.JInternalFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableCellEditor;
import javax.swing.table.TableColumn;

import dk.ruc.blaster.ui.BlasterEditor;

/*
 * dk13043 did not update javadoc, and $Author$ didn't care :)
 *
 * Last change by: $Author$
 *
 * $Header$
 *
 * @version $Revision$
 * @author dk13043
 */
public class PropertyFloat extends JInternalFrame
{
    private static PropertyFloat instance = null;
    JTable table;
    KeyValueTableData data;
    /**
     * Constructor for PropertyFloat.
     */
    private PropertyFloat()
    {
        super();
        this.putClientProperty("JInternalFrame.isPalette", Boolean.TRUE);
        setSize(120, 130);
        setLocation(20, 160);
        data = new KeyValueTableData(null);
        table = new JTable();
        table.setAutoCreateColumnsFromModel(false);
        data.setDefaultData();
        table.setModel(data);

        for (int k = 0; k < KeyValueTableData.m_columns.length; k++)
        {
            DefaultTableCellRenderer renderer = new DefaultTableCellRenderer();
```



```
            renderer.setHorizontalAlignment(
                KeyValueTableData.m_columns[k].m_alignment);

            TableCellEditor editor = null;

            if (k == 1)
            {
                editor = new DefaultCellEditor(new JTextField());
            }

            TableColumn column =
                new TableColumn(
                    k,
                    KeyValueTableData.m_columns[k].m_width,
                    renderer,
                    editor);
            table.addColumn(column);

        }

        JTableHeader header = table.getTableHeader();
        header.setUpdateTableInRealTime(true);

        JScrollPane ps = new JScrollPane();
        ps.getViewport().add(table);

        this.getContentPane().add(ps);

        this.setVisible(true);

    }
    public void setData(HashMap map)
    {
        System.err.println("huzaaa" + map);
        data = new KeyValueTableData(map);
        data.setDefaultData();
        table.setModel(data);

    }
    public static PropertyFloat getInstance()
    {
        if (instance == null)
        {
            instance = new PropertyFloat();
        }
        return instance;
    }
    public void refresh()
    {
    }
}
class KeyValueData
{
    public String key;
    public String value;

    public KeyValueData(String key, String value)
    {
        this.key = key;
        this.value = value;

    }
}
```

```
class ColumnData
{
    public String m_title;
    public int m_width;
    public int m_alignment;

    public ColumnData(String title, int width, int alignment)
    {
        m_title = title;
        m_width = width;
        m_alignment = alignment;
    }
}
class KeyValueTableData extends AbstractTableModel
{
    static final public ColumnData m_columns[] =
        {
            new ColumnData("Key", 25, JLabel.LEFT),
            new ColumnData("Value", 40, JLabel.LEFT)};

    ArrayList list = new ArrayList();

    HashMap map;

    public KeyValueTableData(HashMap map)
    {
        this.map = map;
    }

    public void setDefaultData()
    {
        list.clear();
        if (map == null)
            return;
        for (Iterator iterator = map.keySet().iterator(); iterator.hasNext();)
        {
            String element = (String) iterator.next();
            String value = (String) map.get(element);
            list.add(new KeyValueData(element, value));
        }

    }

    public int getRowCount()
    {
        return list == null ? 0 : list.size();
    }

    public int getColumnCount()
    {
        return m_columns.length;
    }

    public String getColumnName(int column)
    {
        return m_columns[column].m_title;
    }

    public boolean isCellEditable(int nRow, int nCol)
    {
        if (nCol == 1)
            return true;
```



```
        else
            return false;
    }

    public Object getValueAt(int nRow, int nCol)
    {
        if (nRow < 0 || nRow >= getRowCount())
            return null;
        KeyValueData row = (KeyValueData) list.get(nRow);
        if (nCol == 0)
            return row.key;
        else
            return row.value;
    }

    public String getTitle()
    {
        return "Properties";
    }
    /**
     * @see javax.swing.table.TableModel#setValueAt(Object, int, int)
     */
    public void setValueAt(Object aValue, int rowIndex, int columnIndex)
    {
        list.set(rowIndex, new KeyValueData( ((KeyValueData)list.get(rowIndex)).
key, (String) aValue));
        HashMap hash = new HashMap();

        for (Iterator iter = list.iterator(); iter.hasNext();)
        {
            KeyValueData d = (KeyValueData) iter.next();
            hash.put(d.key, d.value);
        }

        if (BlasterEditor.currentFrame.getCurrentSelection()== null)
            return;
        BlasterEditor.currentFrame.getCurrentSelection().setProperties(hash);
    }

}
```

Appendiks E

Test kilde kode

E.1 Test kilde kode for klient og server

CollisionTest.cpp

Page 1/1

```
#include "CollisionTest.h"

#include "Matrix.h"
#include "Vector.h"
#include "Polygon.h"

using namespace blaster;

const CollisionTest CollisionTest::registerThis;

void CollisionTest::runTest( std::ostream & out )
{
    Vector verts[6];

    verts[0] = Vector( 0, -1 );
    verts[1] = Vector( -1, 1 );
    verts[2] = Vector( 1, 1 );

    verts[3] = Vector( 0, -1 );
    verts[4] = Vector( -1, 1 );
    verts[5] = Vector( 1, 1 );

    Matrix m1;
    Matrix m2;

    Polygon p1( 0, 1, 2, &verts[0], &verts[1], &verts[2] );
    Polygon p2( 3, 4, 5, &verts[3], &verts[4], &verts[5] );

    m1 *= Matrix::translateMatrix( 0, 0, 0 );
    m2 *= Matrix::translateMatrix( 0, 0, 0 );

    p1.transform( m1 );
    p2.transform( m2 );

    assert( p1.collides( &p2 ) == TRUE );

    m1.loadIdentity();
    m2.loadIdentity();

    m1 *= Matrix::translateMatrix( 5, 0, 0 );
    m2 *= Matrix::translateMatrix( 7, 0, 0 );

    p1.transform( m1 );
    p2.transform( m2 );

    assert( p1.collides( &p2 ) == TRUE );

    m2 *= Matrix::rotateZMatrix( M_PI );

    p2.transform( m2 );

    assert( p1.collides( &p2 ) == FALSE );

    out << "CollisionTest completed successfully" << std::endl;
}
```

CollisionTest.h

Page 1/1

```
#ifndef COLLISIONTEST_H
#define COLLISIONTEST_H

#include "TestBase.h"

namespace blaster
{

    class CollisionTest:public TestBase
    {
      public:
        CollisionTest( void ):TestBase(  )
        {
        }

        virtual void runTest( std::ostream & out );

      private:
        static const CollisionTest registerThis;
    };

}

#endif
```

MessageTest.cpp

Page 1/1

```
#include "MessageTest.h"
#include "ClientNetworkController.h"
#include "ServerNetworkController.h"
#include "TestMessage.h"
#include "TestMessageMaker.h"

using namespace blaster;

const MessageTest MessageTest::registerThis;

void MessageTest::runTest( std::ostream & out )
{
    ClientNetworkController *client =
        ClientNetworkController::getInstance(  );

    ServerNetworkController *server =
        ServerNetworkController::getInstance(  );

    server->startServer( 1337 );

    bool connected = client->connect( "localhost", 1337 );

    assert( connected );

    client->think( 1.0f, 0.0f );
    server->think( 1.0f, 0.0f );

    NetMessage *msg = new TestMessage( 1.0f );

    client->UDPSendMessage( msg );

    server->think( 2.0f, 0.0f );

    assert( TestMessage::recieved == TRUE );

    client->disconnect(  );

    out << "MessageTest completed successfully" << std::endl;
}
```

MessageTest.h

Page 1/1

```
#ifndef MESSAGETEST_H
#define MESSAGETEST_H

#include "TestBase.h"

namespace blaster
{

    class MessageTest:public TestBase
    {
      public:
        MessageTest( void ):TestBase(  )
        {
        }

        virtual void runTest( std::ostream & out );

      private:
        static const MessageTest registerThis;
    };

}

#endif
```

PlayerAddTest.cpp

Page 1/1

```
#include "PlayerAddTest.h"

#include "Player.h"
#include "GameController.h"

using namespace blaster;

const PlayerAddTest PlayerAddTest::registerThis;

void PlayerAddTest::runTest( std::ostream & out )
{
    Player *player = new Player(  );

    player->setID( 42 );

    GameController *game = GameController::getInstance(  );

    game->addPlayer( player );

    assert( player == game->getPlayer( 42 ) );

    game->removePlayer( 42 );

    assert( game->getPlayer( 42 ) == NULL );

    out << "PlayerAddTest completed successfully." << std::endl;
}
```

PlayerAddTest.h

Page 1/1

```
#ifndef PLAYERADDTEST_H
#define PLAYERADDTEST_H

#include "TestBase.h"

namespace blaster
{

    class PlayerAddTest:public TestBase
    {
      public:
        PlayerAddTest( void ):TestBase(  )
        {
        }

        virtual void runTest( std::ostream & out );

      private:
        static const PlayerAddTest registerThis;
    };

}

#endif
```

ShipDeathTest.cpp

Page 1/1

```
#include "ShipDeathTest.h"

#include "Player.h"
#include "Level.h"
#include "Ship.h"
#include "GameController.h"

using namespace blaster;

const ShipDeathTest ShipDeathTest::registerThis;

void ShipDeathTest::runTest( std::ostream & out )
{
    GameController *game = GameController::getInstance(  );

    Player *p = new Player(  );

    p->setID( 42 );

    game->addPlayer( p );

    // Give the player a chance to spawn a ship
    game->think( 1.0f, 0.0f );

    // Now the player ought to have a ship
    assert( p->getShip(  ) != NULL );

    // Now lets take it away from him
    float damage = p->getShip(  )->energyCapacity(  ) + 1.0f;

    // bam!
    p->getShip(  )->takeDamage( damage );

    // Give the ship a chance to remove itself
    game->think( 1.1f, 0.0f );

    // Now there should be no ship
    assert( p->getShip(  ) == NULL );

    // Advance the time below the respawn threshold
    game->think( 1.2f, 0.0f );

    // Still no ship, right?
    assert( p->getShip(  ) == NULL );

    // Let the player respawn a ship
    game->think( 5.0f, 0.0f );

    // Make sure that the respawn works
    assert( p->getShip(  ) != NULL );

    out << "ShipDeathTest completed successfully" << std::endl;

    delete p;
}
```

ShipDeathTest.h

Page 1/1

```
#ifndef SHIPDEATHTEST_H
#define SHIPDEATHTEST_H

#include "TestBase.h"

namespace blaster
{

    class ShipDeathTest:public TestBase
    {
      public:
        ShipDeathTest( void ):TestBase(  )
        {
        }

        virtual void runTest( std::ostream & out );

      private:
        static const ShipDeathTest registerThis;
    };

}

#endif
```

TestBase.cpp

Page 1/1

```
#include "TestBase.h"

using namespace blaster;

Tests TestBase::tests;

TestBase::TestBase( void )
{
    tests.push_back( this );
}

void TestBase::runAllTests( std::ostream & out )
{
    for ( Tests::iterator i = tests.begin(  ); i != tests.end(  ); i++ )
    {
        ( *i )->runTest( out );
    }
}

int main( int argc, char **argv )
{
    TestBase::runAllTests( std::cout );

    return 0;
}
```

TestBase.h

Page 1/1

```
#ifndef TESTBASE_H
#define TESTBASE_H

#include <assert.h>
#include <list>
#include <iostream>
#include <string>

namespace blaster
{

    class TestBase;

    typedef std::list < TestBase * >Tests;

    class TestBase
    {
      public:
        TestBase( void );

        virtual void runTest( std::ostream & out ) = 0;

        static void runAllTests( std::ostream & out );

      private:
        static Tests tests;
    };

}

#endif
```

E.2 Test kilde kode for editor

ExporterTest.java Page 1/2

```
/**
 * ------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------
 */

package dk.ruc.blaster.export;

import java.awt.Point;
import java.io.File;

import dk.ruc.blaster.model.Map;
import dk.ruc.blaster.model.Vertex;

import junit.framework.TestCase;

/**
 * Class to test Export to XML in the {@link Exporter} class
 *
 * Last change by: $Author: dyregod $
 *
 * $Header: /var/cvs/Alwazah\040Editor/test/dk/ruc/blaster/export/ExporterTest.j
ava,v 1.1 2002/12/15 18:40:31 dyregod Exp $
 *
 * @version $Revision: 1.1 $
 * @author dk13043
 */
public class ExporterTest extends TestCase
{

    File file = new File("fisk.xml");
    Map map = new Map();
    /**
     * Constructor for ExporterTest.
     * @param arg0
     */
    public ExporterTest(String arg0)
    {
        super(arg0);
    }

    public static void main(String[] args)
    {
        junit.textui.TestRunner.run(ExporterTest.class);
    }

    /**
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception
    {
        super.setUp();
        Vertex verts[] =
            new Vertex[] {
                map.addVertex(new Point(1, 1)),
                map.addVertex(new Point(5, 1)),
                map.addVertex(new Point(1, 5))};

        map.addTriangle(verts);
```

ExporterTest.java Page 2/2

```
        verts =
            new Vertex[] {
                map.addVertex(new Point(1, 5)),
                map.addVertex(new Point(5, 1)),
                map.addVertex(new Point(5, 5))};

        map.addTriangle(verts);
    }

    /**
     * @see TestCase#tearDown()
     */
    protected void tearDown() throws Exception
    {
        super.tearDown();
        file.delete();
    }

    /**
     * Tests Export method. Will only test if export does
     * not fail. Not if exported file is valid.
     */
    public void testExport()
    {

        try
        {
            Exporter.export(map, file);
            System.out.println(file.exists());
            if (file.length() != 0)
                assertTrue(true);
            else
                assertTrue("File length is 0", false);
        }
        catch (Throwable e)
        {

            assertTrue(e.getMessage(), false);
        }
    }

}
```

MapTest.java Page 1/3

```
/**
 * ------------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------------
 */

package dk.ruc.blaster.model;

import java.awt.Point;
import java.awt.Rectangle;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import junit.framework.TestCase;

/**
 * Used in testing the {@link Map} class
 *
 * Last change by: $Author: dyregod $
 *
 * $Header: /var/cvs/Alwazah\040Editor/test/dk/ruc/blaster/model/MapTest.java,v
1.2 2002/12/19 01:14:38 dyregod Exp $
 *
 * @version $Revision: 1.2 $
 * @author dk13043
 */
public class MapTest extends TestCase
{
    Map map = new Map();
    HashMap hmap = new HashMap();
    Triangle t1;
    Triangle t2;

    /**
     * Constructor for MapTest.
     * @param arg0
     */
    public MapTest(String arg0)
    {
        super(arg0);
    }

    /**
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception
    {
        super.setUp();

        hmap.put("width", String.valueOf(5000.0f));
        hmap.put("height", String.valueOf(5600.0f));
        hmap.put("gravityx", String.valueOf(5.0f));
        hmap.put("gravityy", String.valueOf(5.0f));
        hmap.put("name", "test");
        map.setProperties(hmap);

        Vertex verts[] =
            new Vertex[] {
```

MapTest.java Page 2/3

```
                map.addVertex(new Point(10, 10)),
                map.addVertex(new Point(50, 10)),
                map.addVertex(new Point(10, 50))};

        t1 = map.addTriangle(verts);

        verts =
            new Vertex[] {
                map.addVertex(new Point(10, 50)),
                map.addVertex(new Point(50, 10)),
                map.addVertex(new Point(50, 50))};

        t2 = map.addTriangle(verts);
    }

    public void testBounds()
    {
        assertTrue( (map.bounds().width == 5000.0f) );
        assertTrue( (map.bounds().height == 5600.0f) );
    }

    public void testGetProperties()
    {
        HashMap hmapNew = map.getProperties();

        Iterator iter = hmapNew.keySet().iterator();

        while (iter.hasNext())
        {
            Object obj = iter.next();
            if (!hmapNew.get(obj).equals(hmap.get(obj)))
            {
                assertTrue(((String)obj) + " does not contain the right value", false);
                return;
            }

        }
        assertTrue(true);
    }

    public void testGetVerticesInRect()
    {
        Rectangle rect = new Rectangle(0,0,40,60);
        List list = map.getVerticesInRect(rect);
        assertTrue((list.size() == 2));

        rect = new Rectangle(5,5,10,10);
        list = map.getVerticesInRect(rect);
        assertTrue((list.size() == 1));

        rect = new Rectangle(20,20,20,20);
        list = map.getVerticesInRect(rect);
        assertTrue((list.size() == 0));
    }

    public void testGetTrianglesInRect()
    {
        Rectangle rect = new Rectangle(20,20,20,20);
        List list = map.getTrianglesInRect(rect);
        assertTrue((list.size() == 2));

        rect = new Rectangle(0,0,20,20);
        list = map.getTrianglesInRect(rect);
```

MapTest.java


```
        assertTrue(list.get(0).equals(t1));

        rect = new Rectangle(0,0,5,5);
        list = map.getTrianglesInRect(rect);
        assertTrue((list.size() == 0));
    }

    /*
     * Test for EditorObject select(Point)
     */
    public void testSelectPoint()
    {
        EditorObject obj = map.select(new Point(40,40));
        assertTrue("Object is equal to t2", obj.equals(t2));

        obj = map.select(new Point(20,20));
        assertTrue("Object is equal to t1", obj.equals(t1));

        obj = map.select(new Point(0,0));
        assertTrue("Object is equal to map", (obj.equals(map)) );

        obj = map.select(new Point(11,12));
        assertTrue("Object is an Vertex", (obj instanceof Vertex) );
    }
}
```

VertexTest.java


```
/**
 * ------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------
 */

package dk.ruc.blaster.model;

import junit.framework.TestCase;

/**
 * Class to test {@link Vertex}
 *
 * Last change by: $Author$
 *
 * $Header$
 *
 * @version $Revision$
 * @author dyregod
 */
public class VertexTest extends TestCase
{

    Vertex[] va1 = new Vertex[3];
    Vertex[] va2 = new Vertex[3];

    Vertex v1;
    Vertex v2;
    Vertex v3;
    Vertex v4;
    Vertex v5;

    Triangle t1;
    Triangle t2;
    /**
     * Constructor for VertexTest.
     * @param arg0
     */
    public VertexTest(String arg0)
    {
        super(arg0);
    }

    /**
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception
    {
        super.setUp();
        v1 = new Vertex(0, 0);
        v2 = new Vertex(10, 10);
        v3 = new Vertex(10, 0);
        v4 = new Vertex(-10, 0);
        v5 = new Vertex(-10, -10);

        va1[0] = v1;
        va1[1] = v2;
        va1[2] = v3;
```

VertexTest.java Page 2/2

```
        va2[0] = v3;
        va2[1] = v4;
        va2[2] = v5;

        t1 = new Triangle(va1);
        t2 = new Triangle(va2);
    }

    /**
     * @see TestCase#tearDown()
     */
    protected void tearDown() throws Exception
    {
        super.tearDown();
    }

    public void testMove()
    {
        float posx = v2.getPosition().x;
        float posy = v2.getPosition().y;

        v2.move(2, 2);
        assertTrue(v2.position.x == (posx + 2.0f));
        assertTrue(v2.position.y == (posy + 2.0f));

    }

    public void testGetNumberOfAttachedPolygons()
    {

        assertTrue(
            "Number of attached polygons == 2",
            (v3.getNumberOfAttachedPolygons() == 2));
        v3.removePoly(t2);
        assertTrue(
            "Number of attached polygons == 1",
            (v3.getNumberOfAttachedPolygons() == 1));
        v3.addPoly(t2);
        assertTrue(
            "Number of attached polygons == 2",
            (v3.getNumberOfAttachedPolygons() == 2));

    }

}
```

Appendiks F

CD

Indhold

CD'en indeholder følgende:

- Klient og server i binær version til både Mac OS X og Windows
- Editor i binær version
- Kildekoden til alle programmer
- De fleste nødvendige 3. parts API'er
- JavaDoc til editor kildekoden
- Denne rapport som pdf fil
- README fil der fortæller hvordan man compiler sit eget program og hvordan indholdet på CD'en er organiseret.

Brugervejledning

I dette afsnit vil det blive gennemgået, hvordan programmerne skal bruges.

F.1 Brugervejledning til Blaster

Når Blaster er startet og rumskibet er kommet fra på skærmen kan man styre det. Man bruger piletaster til at styre skibet med. Hvis man trykker på "space"tasten affyrer et skud. Ønsker man at forlade spillet skal man trykker på "ESC"knappen

Nedenunder er en oversigt over anvedte taster.

← drejer skibet til venstre.

→ drejer skibet til højre.

↑ accelererer skibet fremad.

SPACE affyrer skud.

ESC Afslutter Blaster

F.2 Brugervejledning til editor

Når editoren er startet har man et par valgmuligheder. Hvis man ønsker at lave en ny bane skal man trykke på "New"Knappen. Ønsker man at redigere en tidligere oprettet bane skal men trykke på "Open"knappen og finde den gemte fil.

Herefter er det muligt at redigere banen. Man kan vælge om man vil oprette nye polygoner eller om man vil flytte/redigere eksisterende polygoner Hvis man vil oprette polygoner skal man søge for at man er i polygon mode. Dette gøres ved at klikke på "P" i den værktøjbælken inde i vinduet. Man oprette polygoner ved at afsætte 3 punkter på skærmen med musen. Når det tredje punkt er afsat bliver punkterne automatisk forbundet til et polygon. Hvis man i stedet vil redigere et eksisterende polygon skal med sørge for at være i move mode. Dette gøres ved at klikke på "M" i den tidligere nævnte værktøjsbjælke. Det er nu muligt, via musen, at hive fat i punkter og kanter på polygoner og flytte rundt på dem. Det er endvidere muligt at ændre et polygons farve. Dette gøres ved at vælge en polygon med musen og hvorefter værdierne for den vil dukke op i status vinduet. Ved at ændre på værdierne for RGB farverne er det muligt at skifte farve på polygonet.

Når man er færdig med at redigere banen kan man vælge at gemme den, eksportere den eller gøre begge dele. Hvis man vil gemme dem skal man klikke på "Save"knappen og herefter vælge en placering på harddisk og et navn til filen. Når man gemmer filen blive den gemt i et binært format så det er muligt rette i den senere. Vælger man i stedet at eksportere den skal man klikke på "Export"-knappen. Dette eksportere banen til XML-format, hvilket gør det muligt at læse den ind i selve Blaster.

Ønsker man at forlade editoren skal man lukke hovedvinduet, da "exit" i "file"-menuen ikke er implementeret endnu.