

LINKÖPINGS UNIVERSITET

TNM094 - MEDIETEKNISKT KANDIDATPROJEKT

Game of Domes

Medlemmar:

Niklas ANDERSSON

Daniel CAMARDA

Johan ELIASSON

Ellen HÄGER

Jesper HÄGERSTRAND

Fredrik JOHNSON

Oscar WESTBERG

Handledare:

Karljohan PALMERIUS

Kund:

Alexander BOCK

19 oktober 2015

Sammanfattning

Syftet med projektet var att med hjälp av agila systemutvecklingsmetoder utveckla ett spel som ska fungera i domteatern på Visualiseringsscenter C i Norrköping. Spelet skulle fokusera på samarbete mellan de olika spelarna. Projektet utfördes i samband med kursen TNM094 - Medietekniskt kandidatprojekt vid Linköping Universitet, 2014.

Agil utveckling användes under projektets gång vilket betydde att arbetet delades upp i korta tidsperioder. Ett kundmöte hölls varannan vecka, där diskuterades vad som gjorts sen förra mötet och vad som skulle göras tills nästa möte.

Ett flertal färdiga bibliotek användes, till exempel *Simple Graphics Cluster Toolkit*, *OpenSceneGraph* och *Bullet Physics Library*. Dessa användes mest och tog stor del av projektet att implementera och lära sig.

Dokumentationen genomfördes kontinuerligt under projektets gång med hjälp av verktyget *Doxxygen*. Koden dokumenterades utifrån funktionalitet och syfte.

Vissa delar av originalspelidén blev borttagna på grund av tidsbrist och/eller mjukvaruproblem. Men de delar som möjliggör implementering av originalidéerna har behållits för att kunna använda dessa vid ett senare tillfälle.

Resultatet är en grundläggande produkt spelbar i domteatern på Visualiseringsscenter C. Slutprodukten uppfyllde inte alla kundens mål men är en solid grund för vidareutveckling. Det finns 3D-modeller, nätverk, en webbapplikation, fysik samt en simpel serverlösning. Många delar av systemet finns som fungerande prototyper men behöver kopplas samman.

Slutsater man kan dra om projektet är bland annat att det faktiskt är möjligt att utveckla spel till dommiljöer, trots att just spelutveckling är väldigt invecklat. Agil utveckling visade sig spela en central del i att någon form av produkt framställts med god kvalité.

Innehåll

Sammanfattning	i
1 Inledning	1
1.1 Bakgrund	1
1.2 Syfte	2
1.3 Frågeställning	2
1.4 Ursprunglig spelidé	2
1.5 Avgränsningar	3
2 Metod	4
2.1 Utvecklingsprocess	4
2.2 Kravhantering	4
2.3 Dokumentation	4
2.4 Versionshantering	5
2.5 Testning	5
3 Redogörelse för arbetet	6
3.1 Systemarkitektur	6
3.2 Fysik	7
3.3 Nätverk	8
3.4 Webbapplikation	9
3.4.1 App	9
3.4.2 Server	10
3.4.3 Kommunikation	10
3.5 Navigation	10
3.6 Projektilhantering	11
3.7 3D-modellering	12
3.7.1 Moderskepp	12
3.7.2 Stridsskepp (<i>Fighter</i>)	12
3.7.3 Hangar	13
3.7.4 Asteroider	13
3.8 Användargränssnitt	14
4 Resultat och Diskussion	15
4.1 Resultat	15
4.2 Diskussion	16
4.3 Framtida arbete	16

5 Slutsatser	17
5.1 Agilt arbete	17
5.2 Systemarkitektur	17
5.3 Testning	17
5.4 Kodgranskning och Refaktorering	17
5.5 Dokumentation	18
5.6 Kundkontakt	18
5.7 Slutprodukten	18
5.8 Frågeställningar	19
A Lista över använda bibliotek	21
B Roller	22
C Planering	23
D Modeller	25

Kapitel 1

Inledning

Visualiseringcenter C använder idag en domteater där bland annat navigation i rymden utförs i programmet *Uniview*¹. Som en följd av att använda en domteater skapas en mer immersiv upplevelse. Detta eftersom användaren omsluts av domen, gentemot att se bilden rakt framför sig som på en vanlig bildskärm. Inspirerad av detta ville kunden använda domteatern på ett liknande sätt till att skapa ett rymdspel. Kundens tanke var att spelet skulle kunna spelas i domteatern men även på vanliga datorer. Spelet skulle vara samarbetsanpassat där ett flertal spelare ska samarbeta för att besegra ett fiendelag som styrs av andra motspelare. Detta genom att tillsammans styra ett rymdskepp, där varje spelare uppfyller en unik roll. Seger uppnås då det ena laget förstör motståndarens skepp. Projektets syfte är att med en agil arbetsmetodik utveckla ett spel som uppfyller kundens krav.

1.1 Bakgrund

Under våren 2014 har tredjeårsstudenterna på Civilingenjörsprogrammet i Medieteknik haft uppdraget att kombinera en stor del av den kunskap som erhållits under utbildningen och utveckla en mjukvara. Denna rapport handlar om projektet där uppdraget var att skapa ett rymdspel för domteatern på Visualiseringcenter C i Norrköping. Att utveckla ett spel för domen uppfattades som ett spännande men samtidigt utmanande uppdrag för gruppen. Detta eftersom ett rymdspel innehåller många aspekter som kräver kunskap inom bland annat programmering, fysik och matematik. På så sätt är det ett bra test på kunskaperna som lärs ut under medieteknikutbildningen.

Kunden som har beställt spelet är doktoranden Alexander Bock på *Scientific Visualization Group* vid Linköpings Universitet. Alexander berättade om sin idé för gruppen om ett rymdsimuleringspel, vad han hade för krav och några spel som gav honom inspirationen för spelidén. Två av de spelen heter *Star Citizen* och *Guns of Icarus Online*.

*Star Citizen*² är ett förstapersons rymdsimuleringspel som är under produktion i dagsläget. Det finns ett enspelarläge (*singleplayer*) där användaren löser enskilda uppdrag samt ett så kallat sandlådeläge som stödjer flera spelare. I sandlådeläget kan spelaren välja olika roller som till exempel pirat, handelsman, legosoldat och prisjägare. Varje karaktär har sitt eget "liv" i spelet där man kan uppradera och anpassa sitt/sina skepp för sin roll i spelet.

*Guns of Icarus Online*³ är ett samarbetsbaserat förstapersonsspel för flera spelare (*multiplayer*) med *steampunk*-tema. Spelet går ut på att spelaren och lagkamrater flyger ett luftskepp och deltar i luftstrider mot motspelare. Varje spelare tillhör en klass: kapten, skytt eller ingenjör där varje klass har tillgång till olika verktyg.

Vid utveckling av ett spel som ska spelas på en domteater uppstår olika svårigheter. En domteater är en halvsfärisk skärm, detta gör att spelet måste utformas på ett särskilt sätt för att visas korrekt på

¹Uniview, 2014-06-05 <<http://sciss.se/uniview.php>>

²Star Citizen, 2014-06-05 <<https://robertsspaceindustries.com/about-the-game>>

³Guns of Icarus Online, 2014-06-05 <<http://gunsoficalrus.com/gameplay/>>

domskärmen. Domteatern i Norrköping som ska användas har sex olika projektorer där bilden från varje enskild projektor sys (*stitched*) ihop för att ge det slutgiltiga resultatet. Vissa tekniker som kan användas på en vanlig skärm fungerar inte på en domskärm. Allmänt används oftast domteatrar för naturvetenskapliga föreställningar, vilket gör uppdraget att utveckla ett spel för domteatrar unikt.

Uniview är ett av de mest populära systemen för visualisering i domteatrar. Det är ett virtuellt universum som liknar en ”sandlåda” som kan utforskas. I sandlådan finns det inte något bestämt som måste göras utan det går att åka och kolla på precis vad man vill.

1.2 Syfte

Syftet med detta arbete var att bygga upp ett spel som har fokus på samarbete, och som även ska så att köra i domteatern på Visualiseringscentret.

De mål som sattes upp utifrån kundens krav var:

- Flera spelare ska kunna samarbeta från separata enheter
- Möjlighet att spela i dom och VR-arena
- Alla roller ska vara icke-triviala och roliga
- Det ska finnas olika skepp, banor och spellägen

Övriga tekniska krav:

- Kunna spela som kapten, skytt eller pilot i domen / VR-arena
- Kunna spela som ingenjör på plattformar som stödjer *HTML5*
- Spelet ska fungera på nyare versioner av *Windows* och *Mac OSX*
- I domen ska spelet kunna spelas i över 30 bildrutor per sekund och med responstider på 150 millisekunder mellan server och klient

1.3 Frågeställning

- Hur ska olika plattformar kopplas ihop för att köra samma spel?
- Hur ska de olika plattformarna användas för att utnyttja deras olikheter och styrkor för att ge den bästa möjliga spelupplevelsen?
- Fungerar det att utveckla ett spel till domteatern?

1.4 Ursprunglig spelidé

Game of Domes - Space Fight är ett samarbetsorienterat rymdspel. Tillsammans ska en grupp spelare kontrollera ett moderskepp där varje spelare har en egen roll att fylla. Några av rollerna är bland annat kapten och ingenjör. Seger uppnås genom att eliminera fiendens moderskepp.

Spelet fokuserar på samarbete mellan olika roller. Ingen roll kommer att ha fullständig information om den pågående matchen och spelarna måste således kommunicera med varandra för att besegra fienden. De roller som spelarna ska kunna välja mellan är kapten, ingenjör, skytt och pilot.

Kapten

Kaptenens primära uppgift är att styra lagets moderskepp. Till sin hjälp har kaptenen en vag överblick över moderskeppets tillstånd samt en radar som visar det närliggande området. Kaptenen sköter skeppets energifördelning. Som exempel kan kaptenen välja att lägga mer energi på motorerna och mindre på sköldarna för att på så vis få skeppet att röra sig snabbare men samtidigt gör skeppet mer sårbart.

Ingenjör

Till skillnad från de andra rollerna kommer ingenjören inte att spela i ett förstapersonsperspektiv utan istället blicka ner på en instrumentpanel där olika delar av skeppet kan kontrolleras. Från gränssnittet kan ingenjören styra sköldarnas fördelning över moderskeppet, reparera skador, samt hantera resurser. Resurserna används till att köpa uppgraderingar samt tillverka ammunition. Andra spelare kan begära uppgraderingar eller mer ammunition, dock är det upp till ingenjören att besluta om det finns tillräckligt med resurser för att genomföra begäran.

Skytt (*Gunner*)

Skytten sköter moderskeppets vapenarsenal. Varje moderskepp kan ha upp till två skyttar. Skytten kommer vara utrustad med en bättre radar än kaptenen och kan bidra genom att markera fiendens skepp så att kaptenen uppmärksammas på dessa. Bland skyttens arsenal finns en långsam plasmakanon som främst är till för att skada sköldarna på fiendens moderskepp, ett laservapen som främst är till för att förstöra fiendens småskepp samt ett begränsat antal målsökande raketer. Plasmakanonen och raketerna har begränsad ammunition medan lasern drar energi som automatisk fylls på. När skytten behöver mer ammunition kan denne lägga en begäran till ingenjören om att mer ammunition behövs. Skytten är den enda som har möjlighet att byta roll under matchen, dock enbart till pilot.

Pilot

Piloterna är de enda som inte har en roll på moderskeppet. Piloterna styr mindre skepp (*fighters*) och deras uppgift är att förgöra fiendens piloter samt leta efter resurser som finns utspridda på banan. När en pilot hittar resurser kan dessa föras tillbaka till moderskeppet där ingenjören kan utnyttja dem.

1.5 Avgränsningar

Under arbetet har ett antal av originalspelidéns krav och spelfunktioner inte kunnat genomföras på grund av flera anledningar. Tiden är det som har begränsat mest men funktionalitet mellan olika bibliotek och operativsystem har också varit ett stort problem.

Ett av de första kraven som kom fram i början av projektet var funktionalitet mellan olika operativsystem; främst prioriterades *Mac OSX* och *Windows*. Det uppstod komplicerade fel mellan operativsystemet *Mac OSX* och C++-biblioteken, och därför valdes att endast köra spelet på *Windows*. Stöd för *Windows* krävs även för domen vilket också bidrog till att det prioriterades.

De flesta av de andra komponenterna i spelet valdes bort på grund av tidsbegränsningar. Ingenjören och piloten har behållits eftersom de ansågs vara en bra kombination av roller att spela tillsammans. Skytten och kaptenen kändes inte lika viktigt för att uppfylla idén om ett samarbetsorienterat rymdspel. Den nya spelidén består av ett spelläge med maximalt fyra lag med två spelare vardera. Den ena spelaren spelar som pilot enligt det som beskrevs i den ursprungliga spelidén medan den andra spelaren får rollen som ingenjör. Det innebar också att det inte behövdes någon modell för moderskeppet.

Kapitel 2

Metod

Under arbetets gång användes en agil utvecklingsmetod för att utnyttja projektets resurser på bästa sätt. Detta arbetssätt gör även projektet flexibelt för att minimera risken att arbetet står stilla vid problem.

2.1 Utvecklingsprocess

Innan projektet drog igång planerades hur spelet skulle se ut och vara uppbyggt. Via tjänsten *Trello*¹ skapades och underhölls en så kallad produktlogg (*product backlog*). Loggen innehöll alla delar av projektet som behövde göras. Mer om hur en produklog kan fungera finns i [1, Kap 3.2]. För att ta reda på den förväntade tidsåtgången för varje del i projektet användes metoden planeringspoker (*Planning Poker*), se bilaga C. Uppgifterna delades då in efter hur lång tid gruppens medlemmar trodde de kommer ta att färdigställa. Efter att gruppen hade diskuterat alla delar av projektet placerades dessa i loggen på *Trello*.

Utifrån delarna delades Projektet in i ett flertal *sprints*. Varje *sprint* varade i två veckor vilken senare följdes upp av ett kundmöte där föregående *sprint* diskuterades och även vad som ska göras i nästa *sprint*. Arbetsuppgifter delades upp mellan gruppmedlemmarna (se bilaga B) men gruppen arbetade oftast tillsammans. Mer om *sprints* i [1, Kap 3.2].

2.2 Kravhantering

Under projektets gång hölls regelbunden kundkontakt. De generella krav som sattes upp av kunden specifierades och delades upp i *stories* och *tasks*. Dessa adderades till produktloggen i *Trello*. Kraven prioriterades så att saker som var viktiga gjordes först. Då ett krav implementerades/utfördes flyttades *tasken* från ”pågående” till ”klar”.

2.3 Dokumentation

Den slutgiltiga koden dokumenterades kontinuerligt med tydliga och välbeskrivande kommentarer i takt med att den skrevs. För att sammanställa dokumentationen användes verktyget *Doxxygen*². *Doxxygen* är ett program som körs efter att alla kommentarer har skrivits för att sammanställa all dokumentation i en samling *HTML*-filer. Detta ger läsaren en betydligt större överblick och gör det lättare att hitta relevant information om systemet till skillnad mot att läsa igenom källkodsfiler. Dokumentationen beskrev vad alla klasser och funktioner gjorde och hur dessa var tänkta att användas. Den kunde därför användas av en utvecklare för att förstå sig på delar av systemet som en annan utvecklare har

¹Trello, 2014-06-05 <<https://trello.com/>>

²Doxxygen, 2014-06-05 <<http://www.stack.nl/~dimitri/doxygen/>>

skrivit. Allt gjordes tillgänglig på en hemsida³ för att alla skulle kunna utnyttja den under projektets gång.

2.4 Versionshantering

För versionhantering användes verktyget *Git*⁴ tillsammans med tjänsten *GitHub*⁵. *Git* är ett versionshanteringsprogram som dokumenterar alla ändringar som görs i programmet och vem som gjort dessa. Med *Git* är det även möjligt att simultant arbeta på flera versioner av programmet, som till exempel kan innehålla olika funktioner. De olika versionerna kan sedan slås ihop till ett sluttgiltigt program. Det är även möjligt att återgå till en tidigare version av systemet om det till exempel skulle uppstå problem med en nyare version.

För att hela gruppen skulle ha tillgång till källkoden användes tjänsten *GitHub*. *GitHub* kan kopplas till *Git* så att alla filer och versioner synkroniseras med deras hemsida.

2.5 Testning

Testning är en väsentlig del av systemutveckling och bör ske löpande under projektets gång. Genom att under varje *sprint* testa funktionalitet kan utvecklarna säkra att produkten som skapas är av hög kvalité och att den möter kundens krav. Testningen för projektet innebär en fullständig kontroll över systemets funktionalitet, att leta efter buggar och även en granskning över systemets källkod. Det bidrar även till minskat slöseri av tid som uppstår då en del av källkoden måste slängas/bearbetas på grund av till exempel en bugg eller problem med implementation andra funktioner. Mer om hur testning går till och nytta med det finns att läsa om i [2, Kap. 9].

³Game of Domes dokumentation, 2014-06-05 <<http://niklasandersson.net/space-fight/>>

⁴Git, 2014-06-05 <<http://git-scm.com>>

⁵GitHub, 2014-06-05 <<https://github.com/>>

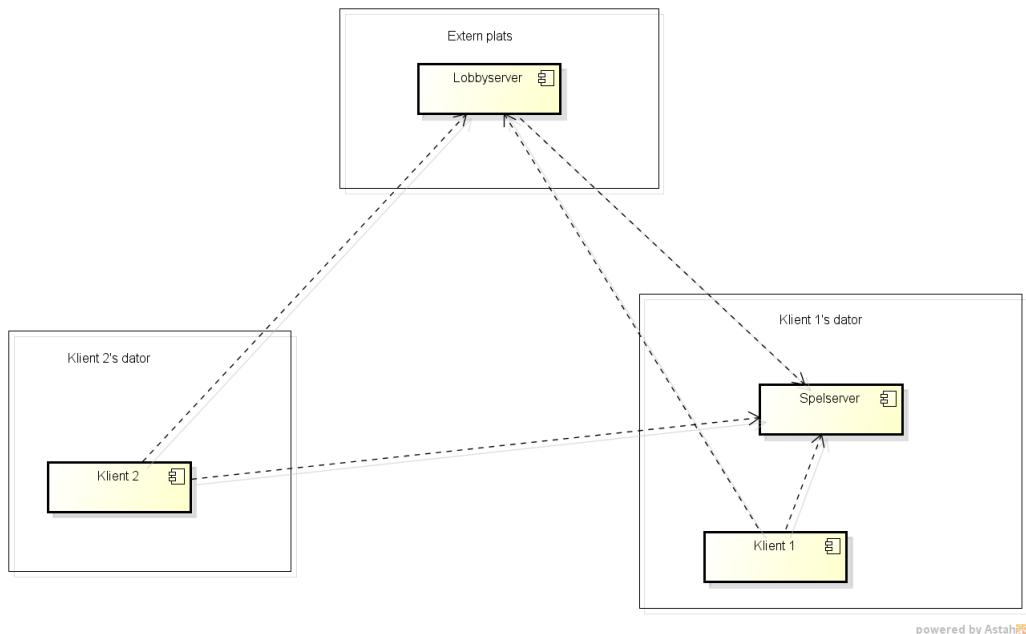
Kapitel 3

Redogörelse för arbetet

Spelet utvecklades för att kunna spelas på olika plattformar, och då även med flera spelare som ansluter via ett nätverk. För att implementera nätverket sker kommunikation mellan applikationerna och en webserver för att tillåta flera spelare och roller samtidigt. Spelet kräver även ett användargränssnitt till piloten och ett till ingenjören samt 3D-modeller.

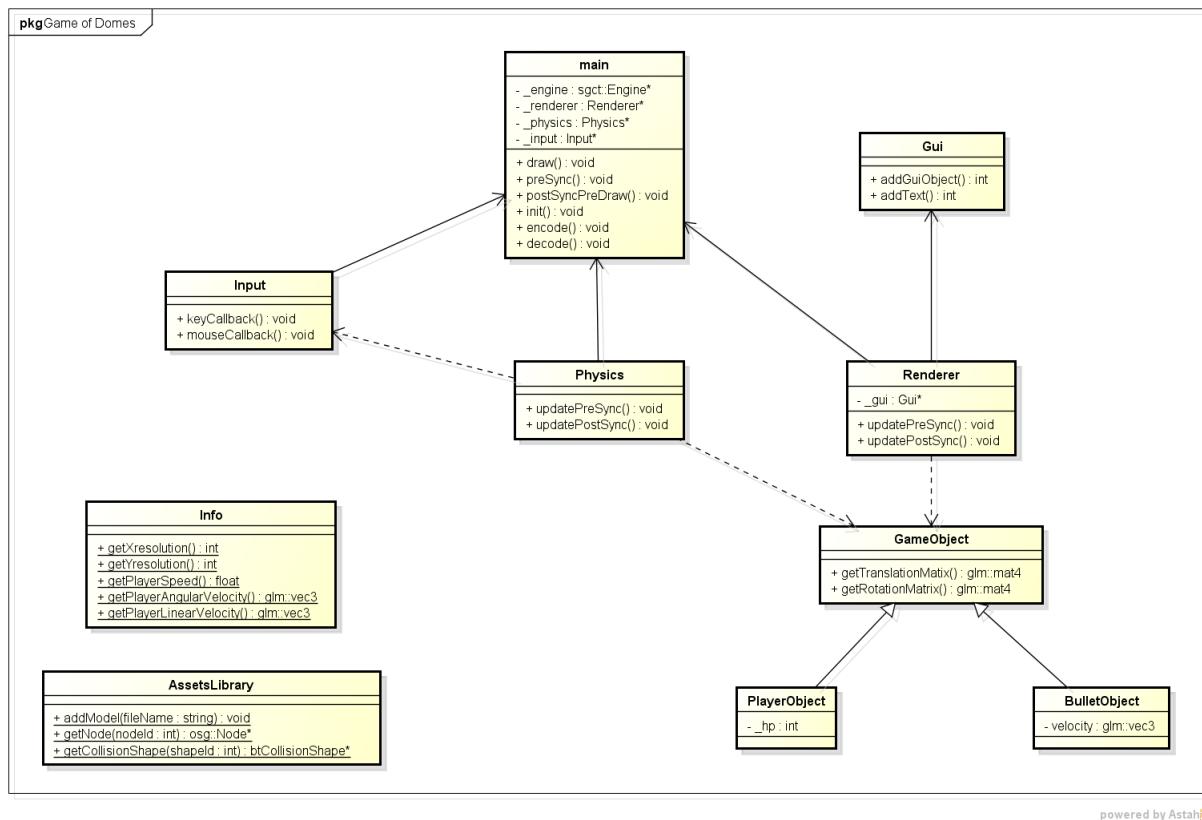
3.1 Systemarkitektur

Systemarkitekturen har skapats stegvis under projektets gång. För att beskriva systemet har allt ifrån whiteboard och papper till modelleringssverktyget *Astah*¹ använts. Syftet med att skapa en systemarkitektur är att alla i utvecklingsgruppen ska förstå hur systemet är tänkt att fungera så att arbetet kan koordineras. Gruppen har främst använt sig av klassdiagram som beskriver kodstrukturen hos systemet och komponentdiagram som beskriver vilka delsystem som finns. Nedan visas ett komponentdiagram, *figur 3.1* och ett klassdiagram, *figur 3.2* som skapades.



Figur 3.1: Komponentdiagram som beskriver de olika delarna av systemet.

¹Astah UML and Modeling Tools, 2014-06-05 <<http://astah.net>>



Figur 3.2: Klassdiagram som beskriver systemet struktur.

3.2 Fysik

Genom att använda en fysikmotor i spelet kunde en mer verklighetstrogen känsla uppnås i spelvärlden. Fysikmotorn som används är *open source*-mjukvaran *Bullet Physics Library*². Med *Bullet* sköts allt ifrån navigation till kollisionshantering på ett fysikaliskt korrekt sätt. *Bullet* är ett C++-bibliotek som innehåller funktioner och klasser för att skapa en dynamisk och interaktiv virtuell fysikvärld. I den här rapporten används benämningen fysikvärlden för att beteckna den virtuella miljö som innehåller fysikaliska representationer av alla objekt och där interaktionen mellan dessa hanteras. Fysikvärlden avbildas sedan till den 3D-scen som visas på användarens skärm, men saknar i sig självt någon direkt koppling till den grafiska representationen av objekten i världen.

För att koppla fysikvärlden i *Bullet* med den grafiska representationen som styrs av *OSG*³ (*OpenSceneGraph*) används ytterligare ett externt bibliotek som heter *osgBullet*⁴. *osgBullet* innehåller en samlingsklasser och funktioner som kan konvertera objekt för användning mellan *OSG* och *Bullet*. Till exempel är det möjligt att med *osgBullet* konvertera en 3D-modell från *OSG* till en så kallad kolisionsform som sedan kan användas för kollisionhantering i *Bullet*.

När spelet startas laddas alla 3D-modeller som ska vara med och sedan konverteras dessa med hjälp av *osgBullet* till fysikobjekt som populerar fysikvärlden. Fysikobjekten är likt 3D-modellerna också polygonytor men en 3D-modell kan innehålla en mycket stor mängd polygoner, som i 3D-scenen ser ut men gör kollisionheteringen i fysikvärlden onödigt långsam. På grund av detta kommer fysikobjekten inte vara exakta representationer av 3D-modellerna utan istället en förenklad version med betydligt färre polygoner.

I *Bullet* skapas alla objekt som så kallade *btRigidBody*-objekt. Varje *rigidbody* kopplas till en

²Bullet Physics Library, 2014-06-05 <<http://bulletphysics.org/>>

³Open Scene Graph, 2014-06-05 <<http://www.openscenegraph.org/>>

⁴osgBullet, 2014-06-05 <<http://osgbullet.vesuite.org/>>

kollsionsform och placeras i fysikvärlden som representeras av ett *btDynamicsWorld*-objekt. Varje *rigidbody* har ett så kallat *motion state* som innehåller en transformationsmatris som ger objektets position och rotation. Innan rendering uppdateras alla 3D-modeller med den transformation som hör till motsvarande fysikobjekt och på så sätt kommer 3D-scenen att efterlikna tillståndet i fysikvärlden.

Fysikvärlden uppdateras innan renderingen körs. Uppdateringen görs genom att stegar simuleringen ett tidssteg framåt. För att simuleringen av fysikvärlden ska bli korrekt krävs det att längden på ett steg i uppdateringen motsvarar den tid som har förflutit sedan den senaste uppdateringen.

Genom att bygga systemet enligt det ovanstående så kommer *Bullet* sedan lösa kollisionshanteringen automatiskt. Fysikmotorn används också till navigationen men där krävs det ytterligare delar för att det ska fungera korrekt.

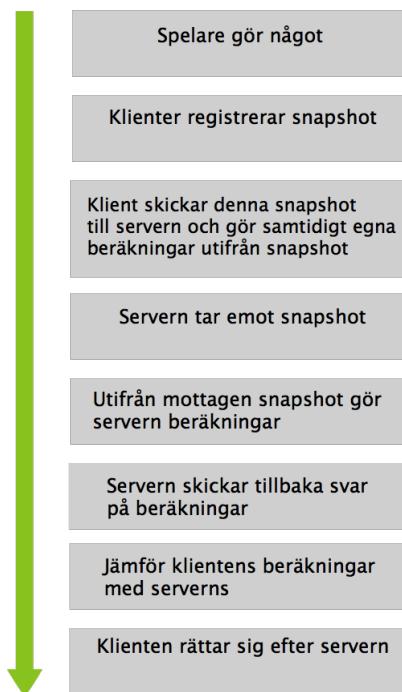
3.3 Nätverk

Servern är hjärtat i spelet och gör att spelet går att spela; det är den som håller ihop alla klienter och ser till att allting står rätt till. Som hjälpmittel för att bygga servern har *ENet*⁵ använts. *ENet* är en *open source* mjukvara som är utformad för att ge en bra grund att börja ifrån vid programmering av nätverksapplikationer i C++. Servern simulerar spelvärlden i steg (*steps*) och för varje steg går servern igenom alla inkommende kommandon som klienterna har angivit. Därefter beräknar servern vad som händer fysikaliskt, kollar om de nya resultaten är tillåtna, och uppdaterar därefter alla objekt i världen. Efter steget har utförts bestämmer servern om någon klient behöver uppdateras och sparar då en *snapshot* av världen i det ögonblicket. Anledningen till varför servern tar hand om allt och inte klienterna själva är för att förhindra fusk.

Man kan inte lita på att varje klient som skickar träff till serven har rätt och sedan acceptera detta. Därför får klienter enbart skicka *snapshots* till serven. En *snapshot* består utav en bild av vad klienten gör i en *frame* (skjuter, rör sig m.m). Utifrån informationen i denna *snapshot* räknar servern ut förändringar till nästa *frame* och skickar ut det till alla klienter.

Då responsiden kan variera mellan olika klienter så använder vi oss av ”*input prediction*”, som betyder att alla klienter simulerar sin egna fysikvärld. Denna fysikvärld gör exakt samma beräkningar som servern gör fast med ett viktigt undantag, den tar bara med det som sker lokalt för din maskin. Detta innebär att varje klient kan simulerar vad som händer lokalt utan att de behöver vänta på svar från servern. När servern har gjort sina beräkningar för serverns fysikvärld skickar den informationen till alla klienter. Klienterna jämför då sina fysikvärldar med serverns och sedan korrigera efter det server säger. I figur 3.3 nedan visas ett händelseförlopp över hur kommunikationen går till. Dessa metoder beskrivs i mer detalj av *Valve Developer Community*[3].

⁵ENet, 2014-06-05 <<http://enet.bespin.org/>>



Figur 3.3: Händelsediagram över klient-server kommunikation

3.4 Webbapplikation

Ett av målen var att spelare ska kunna välja mellan olika roller och sitta på olika platfformar, detta implementerades via en ingenjörsroll spelbar via surfplattor. För att göra detta spelläge så tillgängligt som möjligt programmerades det i *HTML5* och kan därmed användas i alla moderna webbläsare. Resultatet kan ses i *figur 3.4*.



Figur 3.4: Gränssnitt för webbapplikationen

3.4.1 App

Menynerna som ses i *figur 3.4.* är *HTML*-definierade *tables* och får sitt innehåll genom att hämta det från en tillhörande server. Reglagen på sidan av skeppet styr hur sköldarna är fördelade. Dessa reg-

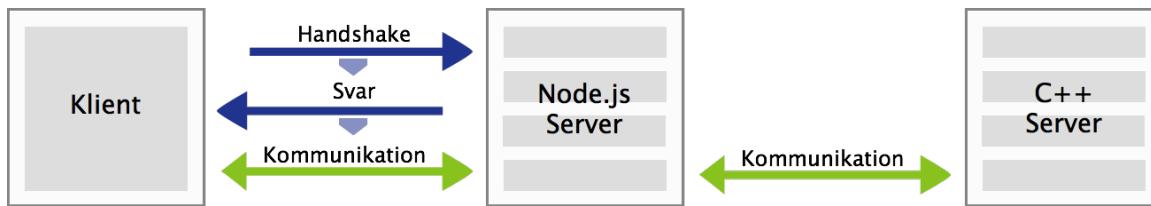
lage är baserade på *JavaScript*-biblioteket *noUiSlider*⁶ och implementeras via *JavaScript*-biblioteket *jQuery*⁷. Reglagen fördelar 100 enheter sköld jämt och tar deras korrelation i åtanke. Vid varje ändring av sköldar, när en användare reparererar, eller köper något skickas ett kommando till en server.

3.4.2 Server

Servern som webbklienten kommunicerar med är skriven i *Node.js*⁸ som i sin tur kommunicerar med en server skriven i C++ som använder *SFML*⁹ för närväck. Mellansteget i *Node.js* krävdes eftersom servern i C++ inte kan hantera direkt kommunikation med en webbklient. Från C++-servern är det lätt att kalla på funktioner i huvudprogrammet och på så sätt är en informationskanal etablerad mellan webbklient och huvudprogrammet.

3.4.3 Kommunikation

Webbklienten initierar en uppkoppling till *Node.js*-servern via ett knapptryck. Klienten skickar en *websocket*, en sorts paket av data standardiserat i *HTML5*, med information om klienten samt tillhörande nyckel. Servern i *Node.js* tar emot paketet och skickar tillbaka ett nytt paket med annan information, här skickas även en ny nyckel baserad på klientens nyckel. Detta informationsutbyte kallas ett *handshake* och tillåter därefter öppen kommunikation mellan klient och server. När uppkopplingen etablerats börjar servern i *Node.js* skicka datapaket till servern i C++ utan att först behöva göra ett *handshake*. På så sätt skapas ett flöde mellan webbläsare och programmet. Processen illustreras i figur 3.5.



Figur 3.5: *Serverkommunikation*

3.5 Navigation

Till en början gjordes navigationen med hjälp av *SGCT*¹⁰ (*SimpleGraphicClusterToolkit*) och *GLM*¹¹ (*OpenGLMathematics*). Varje tangent på tangentbordet kan programmeras till att göra en enskild uppgift med hjälp av *SGCT*. Genom att programmera fyra olika tangenter kan vyn i scenen flyttas framåt, bakåt, vänster och höger. Detta utförs genom olika vektormultiplikationer beroende på vilken riktning vyn ska flyttas.

I rymden sker inte navigeringen enbart i ett plan. För att kunna styra i en 3D-scen används även musen. Med hjälp av *SGCT* kan muspositionen hämtas och sättas. Vid varje *frame* flyttas musens position till mitten av det aktiva fönstret. Om musen har flyttats till nästa *frame* sparas musens position och två vinklar beräknas; en för y-axeln och en för x-axeln. Med hjälp av dessa vinklar kan en 4x4-matris beräknas, vilken senare används för att rotera vyn.

⁶noUiSlider: jQuery Range Slider, 2014-06-05 <<http://refreshless.com/nouislider/>>

⁷jQuery, 2014-06-05 <<http://jquery.com/>>

⁸Node.js, 2014-06-05 <<http://nodejs.org/>>

⁹SFML, 2014-06-05 <<http://www.sfml-dev.org/>>

¹⁰SGCT, 2014-06-05 <<http://webstaff.itn.liu.se/miran/sgct/docs/html/>>

¹¹GLM, 2014-06-05 <<http://glm.g-truc.net/0.9.5/index.html>>

Alla translationer och rotationer beräknades i början av projektet på kamerans position istället för scenens position. Detta medförde att navigationen inte blev korrekt för *fisheye*, det lösades med att matrisoperationerna applicerades på scenen istället för på kameran.

För att simulera hur det skulle vara att styra ett rymdskepp i rymden användes senare spelets fysikmotor för navigationen. Genom att applicera krafter på olika delar av skeppet gav det resultaten att skeppet skulle drivas av ett antal motorer.

Förflyttning av skeppet framåt skedde genom att applicera en centralkraft riktad åt samma håll som skeppet. För att svänga lades två motriktade krafter på skeppet, en längst fram och en längst bak. För att underlätta styrningen implementerades ett kontrollsysteem baserat på en P-regulator. Kontrollsysteemet fungerar genom att användaren får bestämma vilken hastighet denne vill ha och i vilken riktning skeppet ska åka. Sedan avläses skeppets nuvarande riktning och hastighet och skillnaden mellan den nuvarande hastigheten och den önskade hastigheten beräknas enligt ekvation 3.1 nedan där \bar{y} är den nuvarande hastigheten och \bar{r} är den önskade hastigheten.

$$\bar{u} = \bar{y} - \bar{r} \quad (3.1)$$

Resultatet, \bar{u} , från ekvationen används sedan som den kraft som styr skeppet.

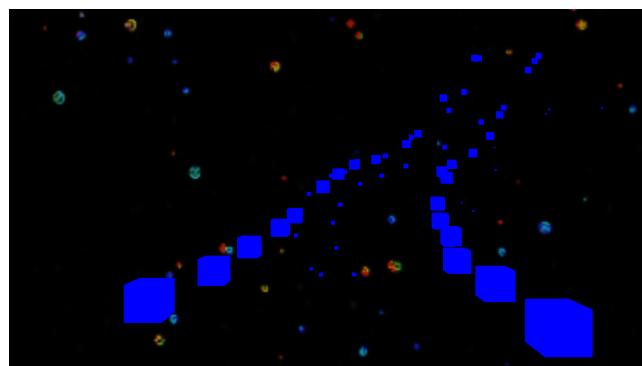
3.6 Projektilhantering

Målet i detta spel är att skjuta ner motståndarna. Med hjälp av *SGCT* kan projektiler skjutas iväg då spelaren klickar med musen. Projektiler skjuts iväg vid varje knapptryckning eller med ett bestämt tidsintervall då musknappen hålls in.

När en projektil skapas behöver flera beräkningar utföras. Projektilen skapas med hjälp av *OSG*, sedan translateras projektilen till den position spelaren siktat på. Då projektilen inte är symmetrisk måste den roteras utifrån vyn så den pekar framåt. Utöver att placera projektilen korrekt måste varje projektil flyttas framåt för varje *frame*. För att kunna göra detta måste positionsvektor, vyriktningsvektorn och vyriktningsmatrisen sparas då projektilen skapas.

För att scenen inte ska fyllas med avfyrade projektiler sparas alla projektiler i en länkad lista (*linked-list*) där de senare tas bort efter en bestämd tid. I och med att projektilerna rör sig framåt kommer de snabbt vara för långt bort för att träffa andra spelare, vilket ger en relativt kort livslängd. I *figur 3.6* syns en tidig version då projektilerna enbart var rektanglar.

Lasern implementerades på samma sätt som projektilerna. Till skillnad från projektilerna är lasern en konstant stråle som har en förutbestämd längd och kommer vara aktiv så länge en tangent är intryckt. För att balansera projektilerna och lasern kommer lasern ta betydligt mindre skada, eftersom lasern är konstant och lättare att träffa med. Lasern skapas vid varje *frame* som vald tangent är intryckt, för att sedan tas bort vid nästa *frame*.



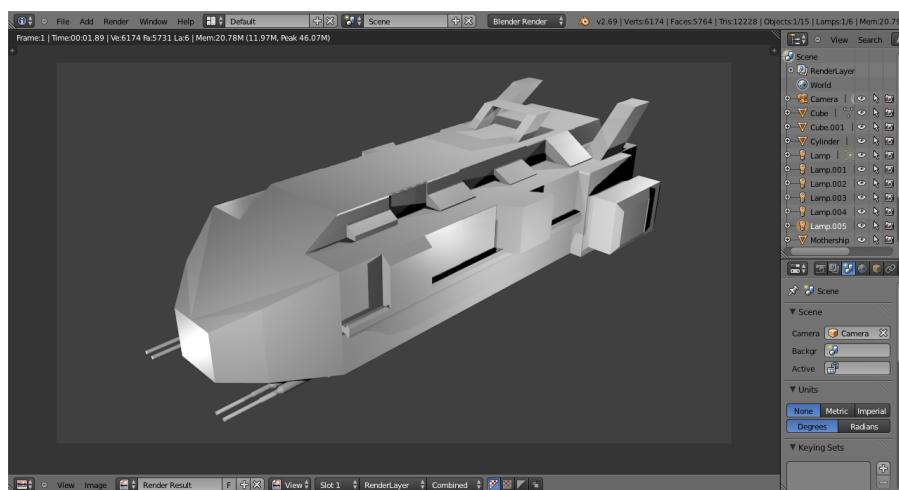
Figur 3.6: Projektiler avfyras

3.7 3D-modellering

Modelleringen skedde i *Blender*¹². *Blender* är *open source*, och allt som skapas är upphovsmannens egendom. Detta betyder att alla modeller som har gjorts tillhör gruppen och får användas hur som helst och till vad som helst. Då texturer är svåra att arbeta med användes *Blenders* inbyggda *material editor* för att ge modellerna egna material för att skapa karaktär och ge objekten trevligare utseende.

3.7.1 Moderskepp

Det första som modellerades var moderskeppet. Det var från moderskeppet som kaptenen skulle styra sitt lag och det var tänkt som det viktigaste skeppet. Moderskeppet utrustades med en stor kanon på undersidan av skeppet, samt mindre kanoner på sidorna som styrs av skytten/skyttarna. Skyttens kanoner modellerades som separata objekt och placerades på höger respektive vänster sida av moderskeppet. Skytten ska även ha möjlighet att avfyra raketer, dessa modellerades som separata objekt som ska kunna skjutas iväg mot fienden. Moderskeppet i *Blender* kan ses i figur 3.7.



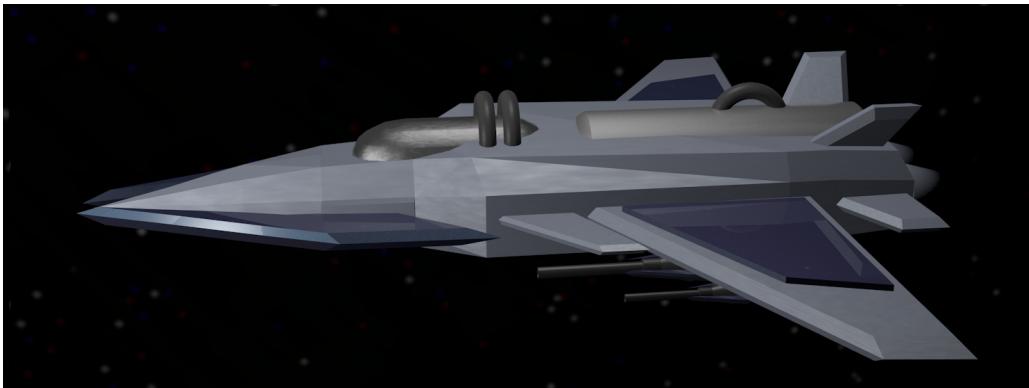
Figur 3.7: Moderskeppet i *Blender*

Fler bilder på moderskeppet finns i bilaga D.

3.7.2 Stridsskepp (*Fighter*)

Stridsskeppet är ett mindre skepp och är endast till för en spelare (pilot). Skeppet modellerades med två små plasmakanoner samt två laserkanoner, vilka placerades under vingarna. Stridsskeppet utformades med en snabbare och smidigare design än moderskeppet, för att ge känslan av att de kan behöva flyga i en atmosfär eller dylikt där luftmotstånd kan påverka flygningen. Materialen på skeppet skapades med metallaktiga egenskaper i åtanke. Största delen av skeppet är i detta material, där kanonerna fick ett mörkare material.

¹²Blender, 2014-06-05 < <http://www.blender.org/> >

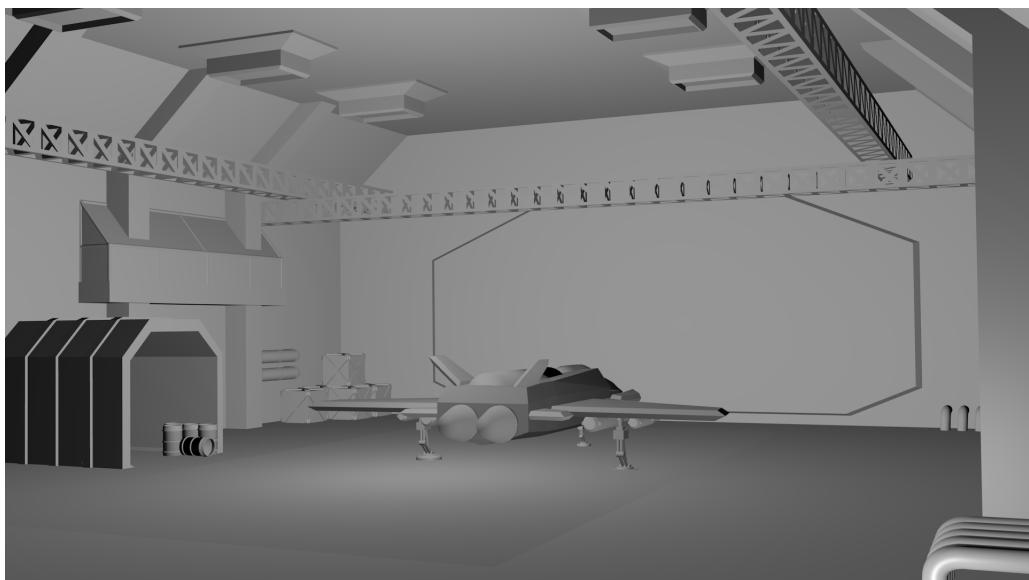


Figur 3.8: Stridsskepp med blå lagfärger

I figur 3.8 syns ett par blå områden, och detta ska föreställa skeppets lagfärger. Dessa varierar beroende på lag, och kan till exempel vara röda eller gröna istället för blå. De runda slangarna fick ett gummiliknande material, och pilothytten har en glasbubbla av mörkt glas.

3.7.3 Hangar

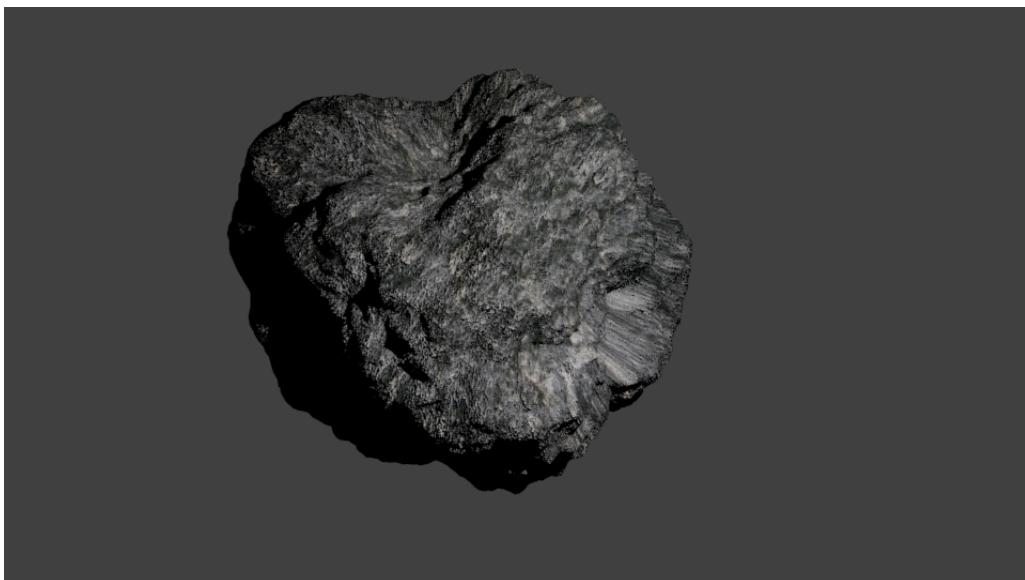
Hangaren modellerades som ett rum där stridsskeppet står och ska används som meny/lobby. Hangaren är ett tomt och öppet rum med en stor port som leder ut i rymden. Det är tänkt att rymma två stycken stridsskepp och återfinns i moderskeppet. En otextruerad bild av hangaren kan ses i figur 3.9.



Figur 3.9: Hangar utan material

3.7.4 Asteroider

Asteroider skapades genom att applicera en *displacement map* på en kub i *Blender*. *Displacement mapping* skapar gropar och förhöjningar i objektet genom att manipulera vektorerna utifrån en höjd-karta (*height map*), som i detta fall var ett slumpräglat genererat moln. Denna modell massproducerades och är tänkt att placeras ut i scenen i grupperingar för att föreställa asteroidbältet och dylikt. Ett exempel på en asteroid kan ses i figur 3.10.



Figur 3.10: Asteroid med material

3.8 Användargränssnitt

Då tre av fyra tänkta roller är i förstapersonsvy används 2D-bilder som användargränssnitt. Metoden med att använda 2D-bilder går ut på att rendera användargränssnittet sist och utan djuptest så att gränssnittet alltid kommer ligga längst fram.

Först skapades skisser för att få en snabb överblick över hur programmet skulle se ut. Dessa användes sedan som evolutionära prototyper och fungerade som grund att jobba vidare på under projektets gång. För att skapa 2D-bilderna användes bildbehandlingsprogrammet *Photoshop*¹³. Genom att använda olika lager går det enkelt att utveckla bilderna med tiden utan att riskera att förstöra gamla versioner.



Figur 3.11: T.V Originalgränssnitt, T.H Gränssnitt kompenserat för fisheye-effekt

För att korrigera *fisheye*-effekten som uppstår när spelet projiceras i en dom används *Photoshops* inbyggda *fisheye*-filter. Filtret appliceras på det färdiga användargränssnittet för att kompensera för den utdragna effekten som ses i figur 3.11.

¹³Photoshop, 2014-06-05 <<http://www.photoshop.com/>>

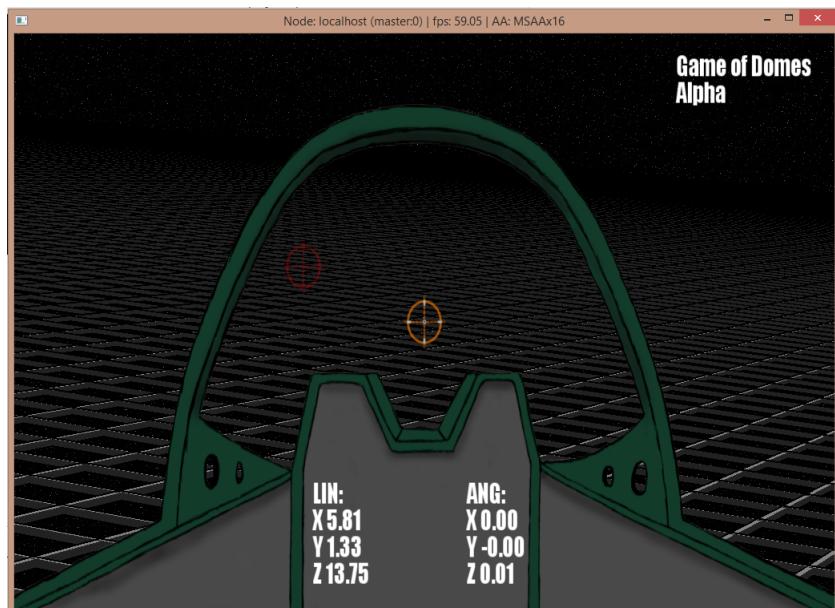
Kapitel 4

Resultat och Diskussion

Många av de tankar och idéer som fanns med från början hann inte komma med i den slutgiltia produkten. Detta på grund av svårigheter med implementation och kommunikation mellan alla olika bibliotek som används. Resultatet blev att produkten nu är en grund till spel och inte ett färdigt spel i sig.

4.1 Resultat

Rollen som pilot går att köra på en PC med *Windows*, medan ingenjör fungerar på alla plattformar som har stöd för *HTML5*. Spelet och webbklienten kommer att ansluta till samma server som fattar alla beslut i spelet. Spelet innehåller endast ett stridsskepp som piloten och ingenjören kommer att befina sig på. Ingenjören har ett 2D-användargränssnitt över skeppet och piloten har en förstapersonsvy från insidan av en cockpit. Det finns även ett moderskepp färdigt att implementera när fler roller har skapats och fler spelare kan ansluta till samma lag. Inga specifika banor är konstruerade utan världen består av en *sky-sphere* och slumpmässigt utplacerade asteroider. Spelet går att spela i både domen och stereo. Nedan i figur 4.1 visas en bild på hur spelet ser ut under körning.



Figur 4.1: Spelet under körning.

4.2 Diskussion

Projektet uppnådde inte alla uppsatta mål på grund av ett flertal olika anledningar. Den främsta anledningen var att de uppsatta målen var överambitiösa. Att lära sig nya bibliotek och verktyg (*OSG*, *SGCT*, *Bullet*, *ENet*, *Blender*, *SFML*, *Node.js*, *HTML5*) och använda dem effektivt tar väldigt lång tid. Dessutom skulle allting implementeras korrekt och spelet skulle göras från grunden.

Fokus för projektet var att få det att fungera i domen vilket krävde *SGCT*. I och med detta krav var vi tvungna att använda *OSG* och få dessa att interagera med en fysikmotor, *Bullet*. Om spelet inte hade behövt vara körbart i domen och då inte behövt använda *SGCT* hade gruppen istället nyttjat en färdig spelmotor. Användningen av en spelmotor hade underlättat arbetet och resulterat i ett mer komplett spel.

Till användargränssnittet användes 2D-bilder. Fördelen med detta, jämfört med att modellera ett 3D-objekt, är att det går snabbt och enkelt att producera många prototyper som senare går att använda i programmet. Detta ger stora fördelar i början av projektet då det snabbt går att få en uppfattning om hur resultatet kommer se ut. Därefter är det enkelt att vidareutveckla prototypen för att använda i den slutgiltiga produkten. En nackdel som uppstår med 2D-bilder är att det är mycket svårt och tidskrävande att tillverka en ”perfekt bild” som ser ut att vara i 3D. Därför är det svårt att få till känslan av att man befinner sig i en 3D-cockpit.

Att utnyttja färdiga bibliotek insågs redan från början behövas. De bibliotek som användes finns listade i bilaga A. Då flera i gruppen tidigare hade testat på grundläggande spelprogrammering stod det klart att det inte skulle vara möjligt att skriva allt själva (rendering- / fysikmotor osv). Vi valde därför att använda oss av ett antal bibliotek i vårt arbete med *Space Wars*, till exempel *SGCT*, *ENet*, *OSG* med flera. Vid första anblick verkar dessa bibliotek alla mycket bra och smidiga att använda. Något som dock ursprungligen inte lades så mycket tanke på är hur dessa interagerar med varandra. På grund av detta har det spenderats mer tid än väntat på att få biblioteken att fungera tillsammans. Biblioteken har dock varit en nödvändig del i att utveckla spelet och som tidigare nämnt var det aldrig ett alternativ att själva utveckla allt.

De *open-source*-bibliotek som har använts har samtidigt haft begränsad dokumentation. Det finns ofta bara klass eller otydliga funktionsnamn i dokumentationen utan en beskrivning om hur de skall användas. Därför har en ”*trial-and-error*” metod behövt använts för att ta reda på hur funktioner skall användas. När alla bibliotek väl fungerade tillsammans, så gick arbetet framåt. I slutändan märks att mer tid har lagts på att få igång biblioteken än att utnyttja dem.

Det var ingen i projektgruppen som hade mer än grundläggande erfarenhet av spelutveckling och det gick inte att återanvända någon tidigare kod, vilket är något som ofta görs av stora spelutvecklare. Spelutvecklare har också i allmänhet längre tid på sig och fler personer är engagerade i utvecklingen.

De medietekniska kunskaper vi har haft användning för är framför allt C++-programmering. Men vi har även haft stor användning av datorgrafik, linjär algebra, reglertechnik och en del fysik.

4.3 Framtida arbete

Projektet har nu fått en stabil grund men är långt ifrån helt klart och kan ännu inte klassas som ett spel. Den tiden som är kvar av kursen kommer framför allt spenderas på att slutföra *gameplay* och nätverkskommunikation så att ingenjören kan kommunicera med piloten. Om dessa delar kan implementeras kommer projektet ha resulterat i ett spelbart spel även om ursprungsmålen fortfarande är avlägsna.

Det har kommit på tal att detta projekt eventuellt ska fortsätta utvecklas nästa år. Inför ett sådant arbete kommer vi försöka lämna ett lättförståeligt och väldokumenterat system.

Kapitel 5

Slutsatser

5.1 Agilt arbete

Att jobba agilt har visat sig vara ett effektivt sätt att utveckla programvara dynamiskt. Då ett projekt kontinuerligt förändras och anpassas hjälper *sprints* och retrospekt till att programmet går åt rätt håll. Vi har mött flera hinder under projektets gång som innebar att vi hamnade efter vår ursprungliga planering, men sprintmöten innebar att vi gjorde det bästa av situationen ur kundens perspektiv.

Något vi kunde ha förbättrat var våra retrospekt, att vara tuff mot sig själv under dessa möten hade inneburit att vi lärt oss mer om vår process och utifrån det förbättra den ytterligare. Vi hade även fått bättre koll på projektet om vår *Trello-board* haft större fokus. Mindre och mer detaljerade *tasks* hade gjort det lättare att välja arbetsområde och ingett en känsla av framsteg.

5.2 Systemarkitektur

Modellering och systemarkitektur har varit en viktig del av projektet. När det använts har det varit en bidragande faktor till att få alla i gruppen att förstå hur systemet ska vara uppbyggt och hur målen uppnås. Det har dock varit svårt att skapa detaljerade modeller och diagram då en stor osäkerhet funits kring hur de använda biblioteken fungerade ihop. Resultatet blev att modellering skedde när gruppen ansåg det nödvändigt, vilket var främst efter att gruppen fick förståelse i hur biblioteken fungerade.

5.3 Testning

Att testa funktionaliteten hos programmet föll lätt genom stolarna trots att det är en central del i agil utveckling. Grunden i vårt program tog så pass mycket tid att genomgående testning aldrig blev aktuellt. Istället blev testningen mer trial and error då vi ville hitta lösningar för att ta oss vidare i utvecklingen och inte så mycket för att hitta dolda fel och buggar. I framtida projekt vore det bra att planera in testning i sprinterna. Att avsätta tid åt testning ser till att programmet håller hög kvalité genom hela projektet och sparar tid vid felsökning ju större programmet blir.

5.4 Kodgranskning och Refaktorering

Kodgranskningen var den del av arbetssättet som var svårast att komma igång med på allvar. Det berodde främst på att det mestadels utvecklades prototyper i början av projektet som gruppen inte ansåg vara värd att granska. I takt med att projektet närmat sig sitt slut har kodgranskning planerats in som ett sista steg för att säkerställa kodens kvalité. Det hade varit bättre att kontinuerligt granska

vår kod. Varje *task* skulle ha innehållit en kodgranskningsdel så att all kod behövt granskas innan en *task* kan klassificeras som klar.

Kodrefaktoreringen underlättade fortsatt användning och underhåll av programmet. När fler och fler moduler sattes samman gick det relativt smärtfritt då koden blev lättläst och logiskt strukturerad. Programmet blev också mer lättförståligt för mindre insatta att förstå.

5.5 Dokumentation

Dokumentering av koden i takt med att den skrevs visade sig mycket smidigt och det är troligtvis det bästa sättet att se till att det blir gjort. Att dokumentationen sedan gjordes tillgängligt på nätet så alla i gruppen kunde ta del av den ansågs bra, men eftersom arbetet ofta skedde tillsammans så användes den aldrig till sin fulla potential.

Däremot för efterkommande utvecklare eller för folk som vill bara vill göra projekt till domer så kan dokumentationen användas istället för den minimalistiskt existerande som vi själva använt.

5.6 Kundkontakt

Kundmöten har spelat en stor roll i projektet. Att involvera kunden har lett till att vi alltid varit på rätt väg med projektet. Att vi hade en tekniskt kompetent och välsinnad kund är något man inte alltid kan räkna med i arbetslivet. När vi stötte på problem var vår kund förstående och gav oss ofta fria tyglar.

För att förbereda oss inför mindre samarbetsvilliga kunder i framtiden skulle vi haft med vår kund mer i utvecklingen och planerat *sprinterna* tillsammans. Att även ordentligt förklara och呈现出 vår *Trello board* varje kundmöte hade varit bra.

Vi hade inget kundkontrakt då det kändes onödigt för ett skolprojekt, i arbetslivet är detta dock en nödvändighet. Som det var nu hände inget om vi halkade efter i planeringen vilket man aldrig kan räkna med i riktiga arbetslivet.

5.7 Slutprodukten

Slutresultatet blev en grundläggande spelmotor för framtida utvecklande av spel till domen. Gruppen var inte tillräckligt stor, och tiden räckte helt enkelt inte till för att åstakomma särskilt mycket mer. Arbetsbördan blev för stor för att hinnas med under kurserna.

Att utveckla ett spel till domen har varit mycket omständligt. Då det inte har funnits några tidigare spel att utgå ifrån har allt byggts från grunden. Det tog oerhört lång tid att bygga en ny spelmotor som går att köra med i domen, och det blev tyvärr ganska lite fokus på sjävla spelet. Idén med att skapa ett spel till en domteater är bra, men det krävs mycket jobb för att skapa rätt typ av spel och för få rätt känsla med samarbete. Även om spelet inte är komplett vid dagens läge är det visat att spelutveckling till domteatrar är möjligt.

För att kunna köra spelet på vilken plattform som helst skapades det en roll som spelas via en 2D-vy och då kunde HTML5 används då det inte kräver 3D-grafik. Detta var en bra lösning på det krav som kunden ställde, då det gör så att många fler plattformar blir tillgängliga.

Nätverket är nästan klart för implementering med server, detta skulle då möjliggöra integration av flera spelare från olika plattformar till samma spelvärld. Fysiken i världen betedde sig inte alltid som den skulle. Detta löstes till viss del med hjälp av ett reglersystem.

Användargränssnittet till piloten (3D-vyn) valdes till en 2D-bild. Detta var en snabb och tillräckligt bra lösning, men det hade varit att föredra att ha en 3D-modell. Detta hade gett mer liv och kanske bidragit till integration med världen. Som det ser ut nu är gränssnittet statiskt.

För att ytterligare öka spelkänslan önskades möjlighet för spelarna att kunna skjuta och förstöra varandras skepp. Detta skulle då gått med att införa livmäta samt sköldar. Möjlighet att eventuellt kunna ha flera olika skott att skjuta med hade också förbättra spelupplevelsen.

5.8 Frågeställningar

- Hur ska olika plattformar kopplas ihop för att köra samma spel?

I *figur 3.1* visas den server/klient-struktur som möjliggör kommunikation mellan alla de plattformar som spelet körs på. På grund av att olika typer av plattformar var inblandade innebar det att olika bibliotek behövde användas. Detta orsakade dock inga problem så länge kommunikationen styrdes av samma protokoll. För att göra detta möjligt användes en extra *Node.js*-server som mellanhand mellan spelklienten och spelservern. Detta beskrivs i *kapitel 3.4.2*.

- Hur ska de olika plattformarna användas för att utnyttja deras olikheter och styrkor för att ge den bästa möjliga spelupplevelsen?

När en användare spelar spelet från Windows-klienten ser denna spelvärlden ur en förstapersonsvy. Det vill säga att spelaren upplever spelet som om denna hade befunnit sig i cockpitén på ett av skeppen. Detta ansågs vara en bra lösning då det ökar inlevelsen i spelet vilket ytterligare stärker känslan av att spela i en dom. En förstapersonsvy ansågs också vara väl lämpad för att styras via mus och tangentbord vilket blev spelets huvudsakliga inmatningsmetoder. I fall framtida arbete hade resulterat i möjligheten att styra med handkontroll så hade även detta fungerat bra.

Den andra plattformen som användes var ett webbgränssnit utvecklad för att kunna användas på en surfplatta eller en mobiltelefon. För den plattformen fattades beslutet att inte skapa någon 3d-miljö överhuvudtaget på grund av att många av de enheter som spelet var tänkt att köras på inte har tillräcklig prestanda för att klara av att rendera avancerade 3d-scener. Det gränssnitt som istället skapades och som syns i *figur 3.4* anpassades för att användas på både stora och små pekskärmar.

- Fungerar det att utveckla ett spel till domteatern?

Med *Game of Domes - Space Fight* var syftet att utveckla ett spel där det utöver möjligheten att spela på en vanlig dator också gick att spela i dom-miljö. Resultatet visar att något sådant är möjligt att genomföra men att mycket extra tid behöver läggas på att få det att fungera jämfört med om spelet hade utvecklats enbart för ett traditionellt datorsystem.

Referenser

- [1] Pekka Abrahamsson, Outi Salo och Jussi Ronkainen, *Agile software development methods: Review and analysis*, VTT 2002
- [2] Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, Pearson Education 2009
- [3] Valve Developer Community: Source Multiplayer Networking, 2014-06-05
https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

Bilaga A

Liste över använda bibliotek

- *SGCT, Simple Graphics Cluster Toolkit* - *SGCT* är ett bibliotek för att programmera interaktiv datorgrafik i dommiljön på Visualiseringsscenter C i Norrköping.
<http://webstaff.itn.liu.se/miran/sgct/docs/html/>
- *OSG, OpenSceneGraph* - *OSG* är en *scene graph* för att enkelt hjälpa till med renderingen.
<http://trac.openscenegraph.org/documentation/OpenSceneGraphReferenceDocs/index.html>
- *Bullet physics Library* - Genom att lägga till objekt med olika fysikaliska egenskaper i en fysikvärld som bullet står för så hanteras alla kollisioner och fysikaliska egenskaper via *Bullet*.
<http://bulletphysics.org/Bullet/BulletFull/index.html>
- *OSGBullet, OpenSceneGraph Bullet* - Eftersom *OSG* är en renderare och bullet simulerar fysiska objekt, så hjälper *OSGBullet* till att konvertera mellan de båda. Man kan läsa in ett objekt i *OSG* och sedan skapas en *mesh* utifrån objektet som *Bullet* sedan kan använda i sin fysikvärld.
<http://vesuite.org/external/docs/osgbullet/>
- *ENet* - *ENet* hjälper till att sätta upp en kommunikation mellan server och klient via *udp*.
<http://enet.bespin.org/index.html>
- *SFML - Simple and Fast Multimedia Library* - Bidrar med flertalet moduler, här används nätverksdelen.
<http://www.sfml-dev.org/>
- *Node.js* - Mjukvaruplattform för nätverksapplikationer.
<http://nodejs.org/>
- *jQuery* - Bibliotek i *JavaScript* som underlättar elementmanipulation.
<http://jquery.com/>
- *noUiSlider* - Bibliotek med fördefinierade reglage i *JavaScript*.
<http://refreshless.com/nouislider/>

Bilaga B

Roller

Niklas Andersson, nikan278 - Dokumentations- och kodgransknings-ansvarig

Arbetade på fysikmotorn och renderingen. Såg till att koden blev ordentligt dokumenterad och att arkitekturen följes.

Daniel Camarda, danca706 - Testningsansvarig

Navigation, fysik och implementation av *OSG* och *Bullet*.

Johan Eliasson, johel964 - Organisationsansvarig

Layout och tillverkning av användargränssnitt.

Ellen Häger, ellha959 - Dokumentansvarig

Arbetade med 3D-modeller. Modellerade alla objekt till spelet och lade till material till dessa.

Jesper Hägerstrand, jesha147 - Produktägare

Arbetade med navigation, projektilhantering samt laserhantering.

Fredrik Johnson, frejo989 - Systemarkitektur

Arbetade med nätverk och att få ihop biblioteken så de fungerar tillsammans.

Oscar Westberg, oscwe917 - Scrum Master

Ansvarade för webbklienten och tillhörande servrar, kundkontakt, möten och mötesprotokoll.

Alexander Bock - Kund

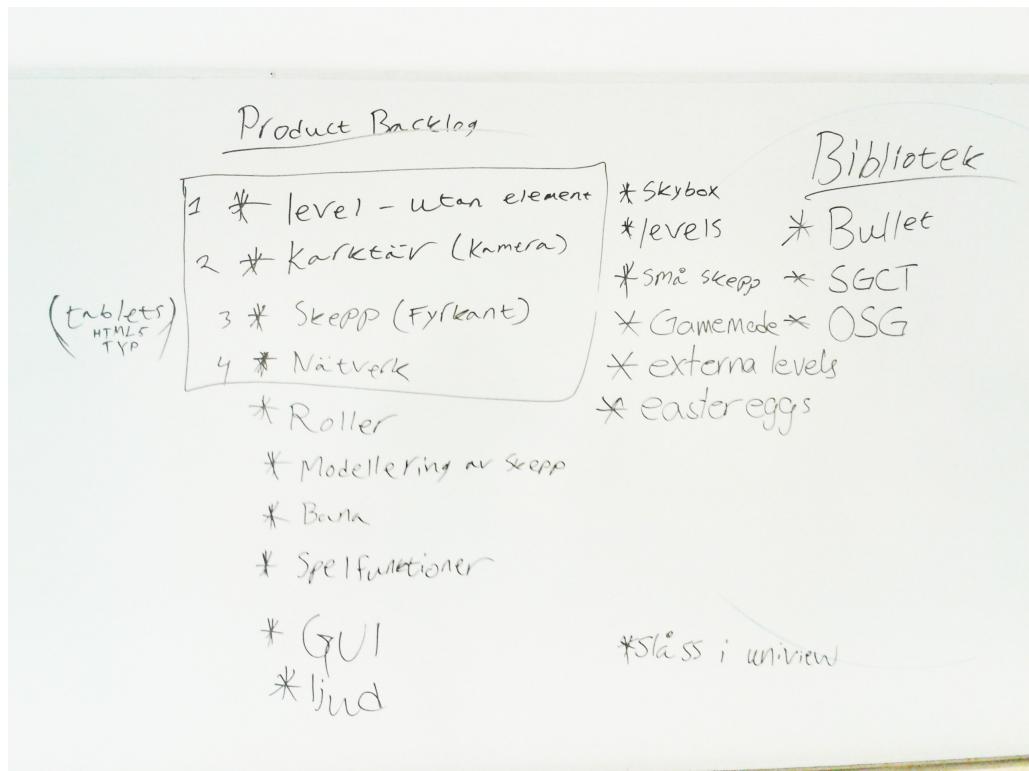
Framförde den ursprungliga projektidén. Har kontinuerligt gett konstruktiv återkoppling under projektet.

Miroslav Andel - Teknisk hjälp

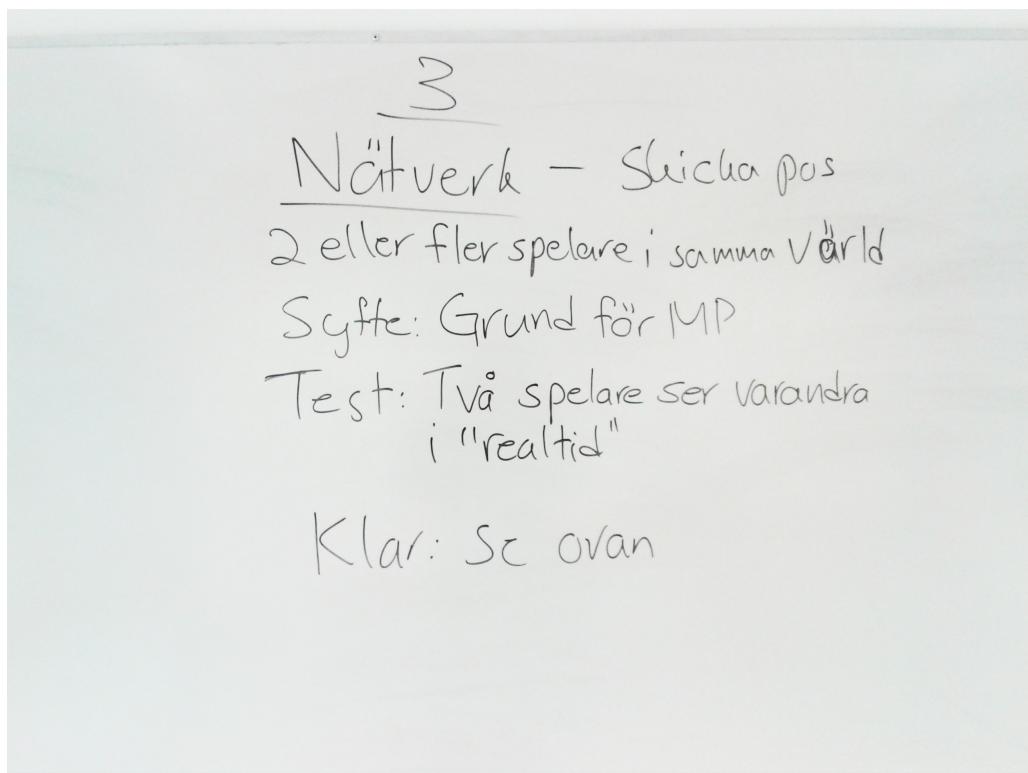
Bidragit med vägledning inom *SGCT*, hjälpte till att projektet kunde påbörjas.

Bilaga C

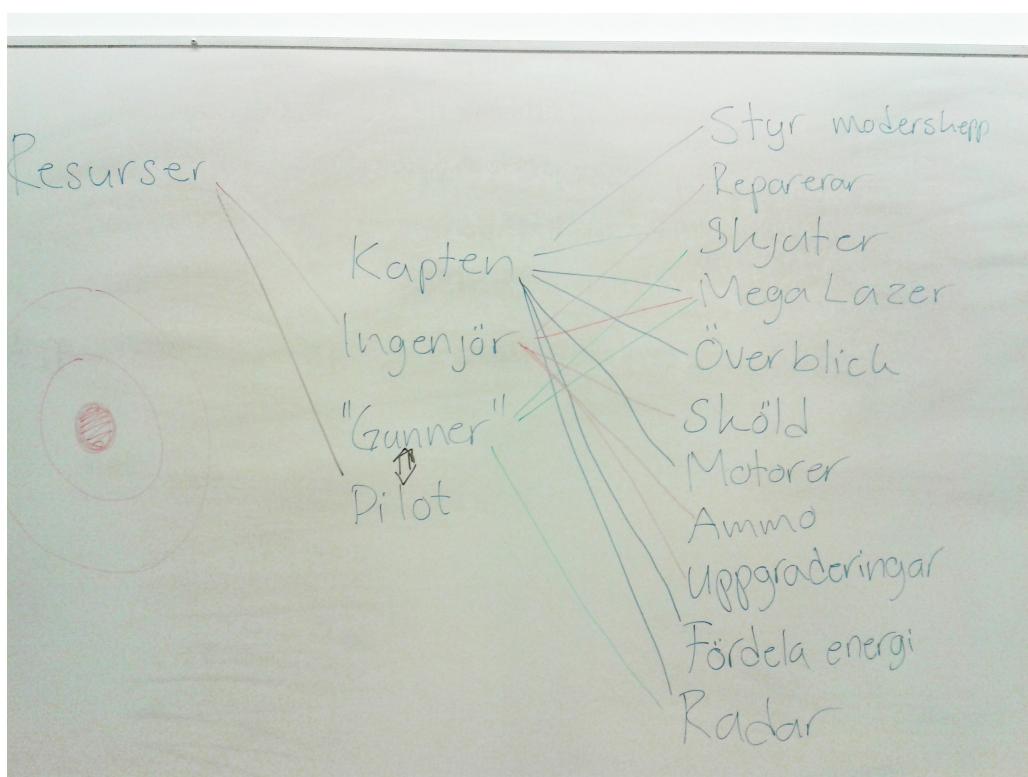
Planering



Figur C.1: Visar en tidig version av produkt backlog & prioritering



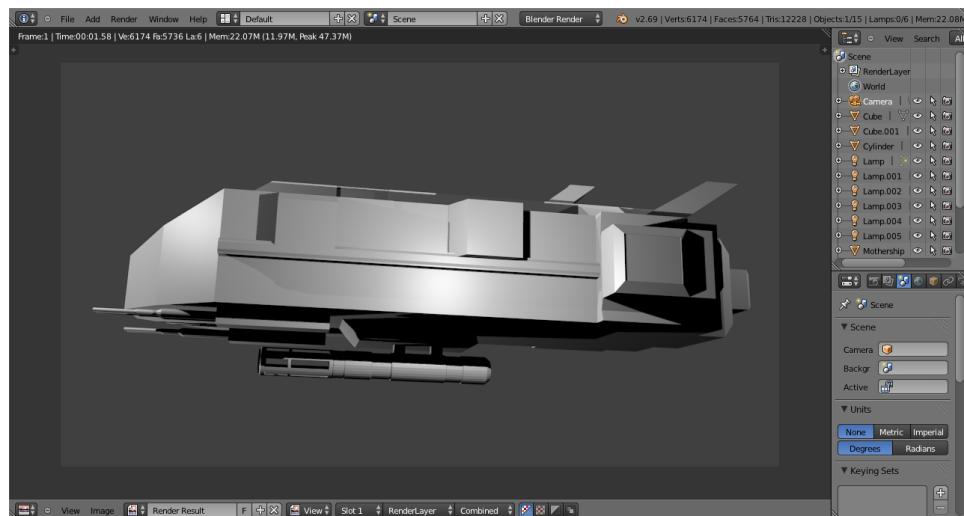
Figur C.2: Planning Poker som visar uppgift och tidsåtgång



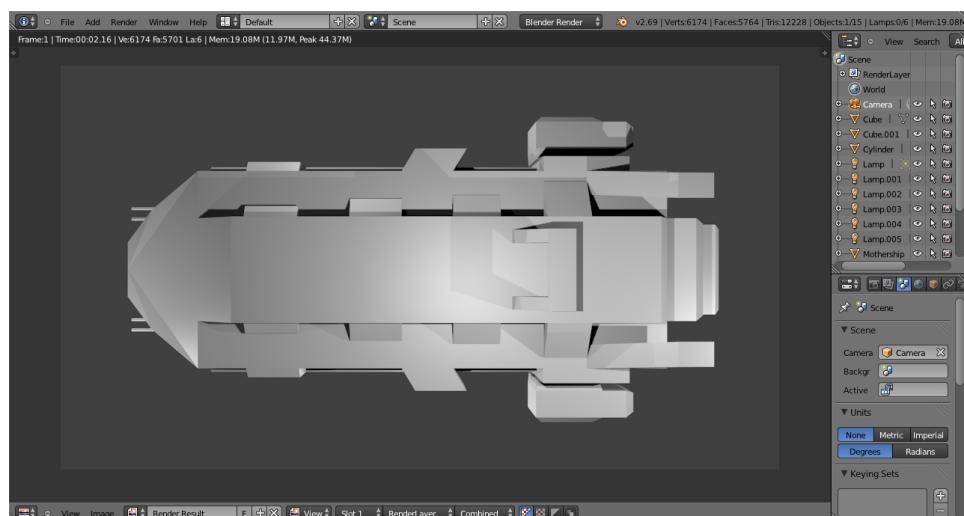
Figur C.3: Vad varje roll skall få göra

Bilaga D

Modeller



Figur D.1: *Moderskepp från sidan*



Figur D.2: *Moderskepp ovanfrån*