Game of Domes: Rymdspel för domteater med mobila enheter
Projektrapport, TNM094

Lars Bergman
Torsten Gustafsson
Max Jonsson
Tobias Pihl
Henrietta Rydfalk

10 juli 2015

Sammanfattning

På uppdrag av Alexander Bock vid Linköpings Universitet skapades en projektplan som följdes för att ta fram ett *multiplayerspel* för en domteater med mobila enheter som kontroller. Spelet utspelar sig i rymden där spelarna samarbetar på ett rymdskepp där de antar olika roller och tillsammans övervinner rymdens faror.

Agil utveckling har använts genom hela arbetet, med mycket kundsamarbete och anpassning. Projektgruppen har bland annat hållit korta, dagliga möten som beskrivs i *scrum*-metoden. Detta har pågått under utvecklingsprocessen för att försäkra att alla medlemmar var medvetna om vilket arbete som utförts. Ansvarsområden har också fördelats. Dessa har innefattat serverutveckling, *OSG*- och *SGCT*-utveckling, grafik och modellering, *Unity*-utveckling samt administration.

Arbetet resulterade i ett *multiplayerspel* där upp till tre spelare ansluter till en *server* och väljer var sin roll för att spela. Nätverksservern och domklienten måste inte nödvändigtvis köras på samma dator. Systemet inkluderar en klientapplikation för webbläsare skriven i *HTML5* och *javascript*, en klientapplikation för spelvyn skriven i *C++* med *SGCT*, *OSG* och *OpenGL* samt en server som hanterar anslutningar via *java*. Spelarna kan välja mellan tre roller: pilot, ingenjör och skytt. Dessa använder mobila enheter som kontroller.

Innehåll

Figurer

Kapitel 1

Inledning

Projektrapportens inledande del beskriver vilka krav och idéer som låg till grund för projektet. Vilka avgränsningar gruppen har tagit beskrivs även här.

1.1 Syfte

Syftet med projektet var att tillämpa agil utvecklingsmetodik för att utveckla ett system utgående från, i projektets början satta, kravspecifikationer och mål baserade på en befintlig kodbas. Systemet i fråga ämnades bli ett nätverksbaserat rymdspel för domteatrar och skulle utgå ifrån kunskaper förvärvade under programmets tre första år.

Syftet med denna rapport är att beskriva det arbete som utförts i kursen Medietekniskt kandidatarbete, *TNM094* vid Linköpings Universitet. I rapporten beskrivs det hur gruppen arbetat under projektets gång och vilka metoder och verktyg som behövts för att erhålla resultatet.

1.2 Bakgrund

Detta projekt utfördes som kandidatarbete av fem studenter på Civilingenjörsprogrammet i Medieteknik, tredje året, på Linköpings Universitet. Kunden för detta projekt var Alexander Bock, doktorand på Linköpings Universitet, Scientific Visualization Group. Uppdraget var att skapa ett rymdrollspel att spelas på domteatern i Visualisieringscenter C i Norrköping.

Föregående år, 2014, utfördes samma projekt med samma kund som kandidatprojekt [1]. Det nuvarande projektet bygger på resultat från det tidigare projektet. Trots att ingenting av den tidigare koden har använts, har lärdomar från den föregående gruppen tagits i beaktning.

1.3 Spelidé

Den ursprungliga spelidén gick ut på att två lag i var sin domteater skulle förstöra motståndarlagets moderskepp. I varje lag skulle det finnas ett antal roller, till exempel ingenjör, skytt, kapten och pilot. Alla roller skulle vara beroende av varandra och huvudsaklig fokus skulle ligga på samarbetet mellan spelarna i ett lag. I diskussion med kund bestämdes det att den spelidén som innefattande två domteatrar inte skulle ha hög prioritet. Istället siktade gruppen på att få ett fungerande spel med endast ett moderskepp där alla roller samarbetar mot ett gemensamt mål.

1.4 Frågeställningar

Detta projekt studerar möjligheterna att utveckla ett nätverksbaserat flerspelarspel för en domteater, potentiellt baserat på en tidigareskriven kodbas för att få en djupare inblick i agil systemutveckling. I denna sektion behandlas frågor ställda med avsikt att avgränsa och tydliggöra vad projektgruppen vill ta reda på under projektets gång.

- När man spelar ett rymdspel i en domteater finns det risk för att man blir åksjuk. Hur ska produkten utvecklas för att undvika detta problem?
- I det här projektet tas kod över från ett tidigare projekt. Kommer tidigare kod kunna användas på ett effektivt sätt, eller är det mer hållbart att skriva kod från grunden?
- Vilka tillvägagångssätt kan antas för att lösa nätverksdelen för ett så pass omfattande projekt?

1.5 Målsättningar

Projektgruppen kom fram till gemensamma mål i början av projektet. Dessa mål är:

- Att besvara de vetenskapliga frågeställningarna
- Att skapa ett spelbart spel med fungerande nätverkskommunikation

1.6 Avgränsningar

Ett antal av projektets ursprungliga idéer prioriterades bort på grund av begränsningar. En typisk avgränsning är målgruppen. Produkten i fråga har inte varit riktad mot någon direkt målgrupp, och därför utvecklats så att den ska vara tilltaglig för en bred publik. Eftersom domteatern får besök av unga såväl som äldre, utvecklas kontroller på ett sådant sätt att det ska vara lätt att förstå för de flesta.

Eftersom spelaren var menad kunna använda sin mobiltelefon som kontroll var det viktigt att tänka på att mobilen har en skärm med begränsad yta. En annan begränsning var att *touch* som input för en mobil enhet gör det svårt att använda för många knappar på samma gång. På grund av detta skapades kontroller på ett sådant sätt att antalet knappar begränsas, samt att de var stora så att det blir svårt att missa dem.

Kunden hade även vissa krav på prestandan i produkten, både i form av bildfrekvens och svarstid, då internetanslutning använts. Detta ledde till två beränsningar: scenen kunde inte ha för mycket grafiskt innehåll och mängden data som skickats över internet var tvunget att begränsas.

1.7 Kundkrav

De enda fasta kraven från kund var att spelet skulle vara körbart i 60 *FPS*, samt att det inte fick vara ”för” lång responstid under körning av spelet. Utöver det har krav diskuterats med kund under projektets gång. Spelet körs utan någon som helst märkbar fördröjning i bildfrekvens och data skickas i realtid.

1.8 Typografiska konventioner

Tekniska och engelska termer markeras med kursiv text. Tekniska ord beskrivs även i sektion 1.9 nedan.

1.9 Beskrivning av förekommande ord

- **SDK**: Software Development Kit
- **TCP**: Transmission Control Protocol, ett protokoll för pålitlig dataöverföring mellan två datorer. Pålitlig i den mån att det garanteras att paket som skickas når destinationen. Genom att skicka med en så kallad kontrollsumma verifieras att storleken på skickade paket vid ankomst inte har ändrats. [2]
- **UDP**: User Datagram Protocol, ett protokoll för opålitlig dataöverföring mellan två datorer. Det finns ingen garanti att paket som skickas når sin slutdestination, fördelen hittas dock i att UDP är mindre resurskrävande och i många fall snabbare. [2]
- **Websockets**: Teknik som möjliggör full-duplex tvåvägskommunikationskanaler över en TCP-anslutning.
- **API**: Application Programming Interface.
- **C++**: Objektorienterat multiplattformsprogrammeringsspråk vidarebyggt på programmeringsspråket C.
- **Java**: Objektorienterat plattformsoberoende programmeringsspråk utvecklat av Sun Microsystems 1995. [3]
- **Javascript**: Skriptspråk för webbapplikationer baserat på ECMAScript.
- **HTML**: HyperText Mark-up Language är ett språk som används till största del som grund för att skriva webbsidor.
- **XML**: Extended Mark-up Language är ett märkspråk på samma sätt som HTML med skillnaden att man kan bygga ut det med egen funktionalitet, vilket gör det användbart inom många tillämpningsområden.
- **GUI**: Graphical User Interface, på svenska grafiskt användargränssnitt, det som användaren ser och interagerar med.
- **Skybox** Den bakgrundsbild som omsluter spelvärlden. Den består av sex bilder, gjorda på ett sådant sätt att de efter sammansättning ger illusionen av att vara en omslutande sfär.
- **Multiplayer** Bokstavligt: *flera spelare*. Beskriver exempelvis ett spel som är anpassat för flera spelare.
- **Mobila enheter** Med mobila enheter syftas här på surfplattor, bärbara datorer och mobiltelefoner.
- **FPS** FPS står för *frames per second*, det vill säga bildrutor per sekund.

Kapitel 2

Relaterat arbete

Då de ursprungliga kundkraven var så pass öppna krävdes en del förarbete kring liknande projekt. Dels för att se på olika tekniker för hur systemet kunde byggas, samt för att finna inspiration till hur olika problem skulle angripas. För att se hur systemet kunde byggas föll det naturligt att betrakta föregående års projekt som även det byggdes för en domteater. I övrigt har ytterligare några projekt undersökts för tankar om spelidé och faktisk spellogik.

2.1 Game of Domes - Föregående projekt

I och med att det här projektet har utförts en gång tidigare vid namnet Game of Domes [1], hämtades mycket inspiration därifrån. Game of Domes lade mycket tid åt biblioteket *bullet*, vilket är ett fysik-bibliotek till *C++*. De försökte även få spelet att bli plattformsoberoende, där framförallt *OSX*-versionen hade visat sig vara problematisk. Dessa tillvägagångsätt prioriterades bort detta år eftersom de inte ansågs vara nödvändiga för att uppnå ett tillfredsställande resultat.

Då föregående gruppen själva hade glömt delar av vad de gjort kunde nästan enbart lärdomar om motgångar tas till vara på. Koden var dokumenterad, men beskrivning av hur projektet skulle startas upp, byggas och kompileras saknades. Detta ledde till att gruppen i slutändan valde att bygga alla bibliotek, och skriva all kod, från grunden. Under processen såg gruppen till att bygga upp egen förståelse för hur systemet fungerar, samt att se till att dokumentation finns för att visa hur projektet sätts upp.

2.2 Guns of Icarus

Guns of Icarus [4] är ett rollspel online för flera spelare som utspelar sig i steampunkmiljö. Detta var en av huvudinspirationerna till spelet, både utifrån kund och projektgrupp. I spelet befinner sig spelarna på ett luftburet skepp där olika roller måste antas; piloter, skyttar och ingenjörer. Spelarna måste sedan samarbeta för att överleva och eliminera fiendeskepp, vilka även de styrs av andra spelare.

2.3 Artemis Spaceship Bridge Simulator

Artemis [5] är ett spel som baserats på liknande idéer som detta projekt. I Artemis simuleras en kommandobrygga på ett rymdskepp genom att koppla samman tre till sex datorer, varav en är huvudskärmen. Alla spelare har varsin roll på bryggan och sin egen arbetsstation för den rollen. Detta med

undantag för kaptenen, som inte har en arbetsstation utan istället bestämmer vad de andra ska göra. Spelet är menat för tre till sex spelare.

Kapitel 3

Systemutveckling

Själva utförandet av arbetet skedde i många steg. Under större delen av tiden har gruppmedlemmar arbetat med olika deluppgifter som kommer beskrivas utförligt i detta kapitel. På grund av gruppens storlek valdes det att arbeta agilt då detta lämpar sig för mindre grupper. Den mall som följdes under projektets gång var *Scrum*.

3.1 Agil utveckling

Inom agil utveckling ligger fokus på dialog och kommunikation för att ta fram ett system eller en produkt. Den agila utvecklingsmetoden följer ett manifest skapat av Agile Alliance 2001 [6]. Detta manifest[7] lyder som följer:

- Individer och interaktioner framför processer och verktyg
- Fungerande programvara framför omfattande dokumentation
- Kundsamarbete framför kontraktsförhandling
- Anpassning till förändring framför att följa en plan

Inom agila metoder arbetar projektgruppen vanligtvis inkrementellt och iterativt. Att arbeta inkrementellt innebär att bryta ner arbetet i mindre uppgifter och bygga projektet del för del, medan iterativ utveckling innebär att delprocesser genom rekursion överses och byggs på flera gånger.

3.2 Planering och administration

Projektet har genomförts med agila utvecklingsmetoder. Till det har arbetsmetoden *Scrum* utnyttjats vilket kommer beskrivas utförligare i detta kapitel. Kortfattat innefattar det att gruppen har hållit korta dagliga möten och arbetat i form av *sprinter* där en fungerande version av spelet har tagits fram efter varje *sprint*, samt vilka rutiner som har använts.

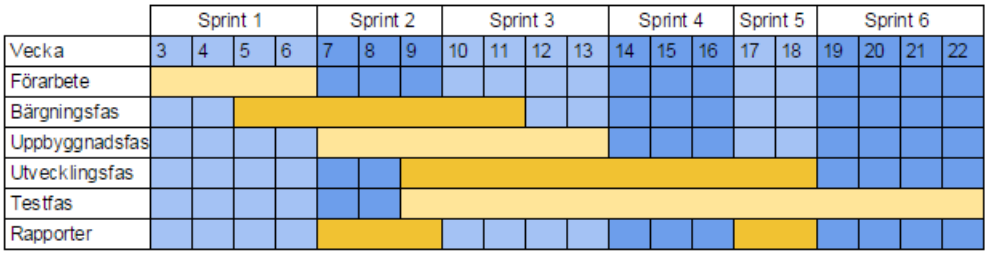
3.3 Scrum

Till detta projekt valdes *Scrum* som utvecklingsmetodik. *Scrum* är en agil utvecklingsmetodik som uppfanns 1994 vid Object Technology samt kommersialiserades av Schwaber och Beedle vid 2002

[6]. Flera självorganiserande och autonoma team utvecklar delar och moduler av produkten parallellt, iterativt och inkrementellt. Utvecklingen sker i perioder om upp till 30 dagar där varje sådan period kallas *sprint*. En *sprint* är en period under vilken ett utvecklingsteam jobbar med ett kortsiktigt mål och i slutet har en del från sprintstart definierade leverabler. Koordinering sker genom så kallade *scrums*. Scrums är korta, avslappnade dagliga möten där dagens mål gås igenom. Arbetet delas upp i *stories* och *tasks*, där *tasks* är uppgifter på detaljnivå och *stories* är större delmoment som görs upp av mindre *tasks*. Vidare finns så kallade *epic stories* som är en större del av projektet som själv består av flera *stories*.

3.3.1 Sprintplanering

Tidigt i projektet skapades en preliminär plan för hur många *sprinter* som skulle täckas, samt hur lång tid dessa skulle ta och vad som skulle implementeras i varje sprint. Den preliminära planen visas i figur 3.1. För att få ytterligare en riktmall i projektet delades de komponenter som var tänkta att implementeras upp i olika faser.



Figur 3.1: Den preliminära sprintplaneringen

Förarbetet innefattade informationsinläsning om projektets olika delar; de *API-er* som kom att användas, vilka tillvägagångssätt som var optimala samt hur mjukvarans arkitektur lämpligen skulle struktureras. Bärgningsfasen var den fas där gruppen studerade den gamla koden. Därifrån bröts den ner och det avgjordes vad som skulle sparas och vad som skulle skrivas om. I uppbyggnadsfasen fokuserade gruppen på att få igång nödvändiga bibliotek och *API:er*. Fasen inkluderade även att få igång nätverksdelen av projektet. Gruppen började i utvecklingsfasen att bygga själva spelet, det vill säga att modeller skapades, funktionalitet implementerades och så vidare. Testning skedde kontinuerligt under utvecklingsfasen.

3.4 Ansvarsroller

För att se till att de saker som skulle göras utfördes korrekt delades gruppens medlemmar in i olika ansvarsroller. Varje roll innefattade ett ansvar för en specifik del av projektarbetet. Rollens innehavare var inte nödvändigtvis den person som utförde det faktiska arbetet, utan såg snarare till att det blev gjort. De roller som bestämdes i början av projektet var följande:

- Server Enchanter, Max Jonsson
- The Wizard of OSG (and SGCT), Torsten Gustafsson
- Scrum Master, Lars Bergman
- Lord of Graphics, Henrietta Rydfalk

- Unity Captain, Tobias Pihl

Alla ansvarsroller hade som uppgift att deras respektive arbetsområden utfördes korrekt. Frågor angående arbetsområdet riktades främst till den ansvarige som även delegerade uppgifter inom området vid behov.

Server Enchanterns roll i gruppen var att hålla koll på vad som fungerade, inte fungerade samt vad som behövde utvecklas för nätverksdelen av systemet.

The Wizard of OSG (and SGCT) hade ansvar för programkod som innefattar *OSG* och *SGCT*. Huvudsakligen *C++* i denna mån.

Scrum master ansvarade för att dagliga *scrums* hölls samt hade ansvar för dokumentation. Denne såg även till att Trello uppdaterades samt hade koll på vad alla i gruppen höll på med.

Lord of Graphics ansvarade för all grafik och dess produktion.

Unity Captain ansvarsroll innefattade utvecklingen av mjukvaran i *Unity*, vilket inkluderade kontroll och logik på mobila enheter.

Dessa roller ändrades under projektets gång då tillvägagångssättet för projektet ändrades. Den största förändringen låg på Unity Captain, som arbetat med serverkod och *GUI* efter att *Unity-utveckling* släpptes helt av gruppen. The Wizard of OSG hade även ett större ansvar för dokumentation tidigt i projektet.

3.5 Möten

Möten har skett dagligen under projekttiden i form av dagliga *scrums*. Dessa möten har hållits korta, cirka 15-20 minuter, och har följt *scrummallen* för hur dagliga *scrums* ska utföras. På grund av att gruppen har läst andra kurser parallellt har arbetsdagarna varit olika fördelade under projektets gång. Under första halvan av projekttiden träffades gruppen tre dagar i veckan, onsdag till fredag, och ägnade dessa dagar helt åt projektarbetet. Under den andra halvan var gruppens sammankomster mer oregelbundna, och kunde innehålla avbrott under dagen för andra kurser. Gruppen strävade dock under perioden efter att träffas varje dag.

3.6 Kundmöten

I projektplanen skrev gruppen att kundmöten ska hållas i *sprinternas* slut. Mötet innefattar att gruppen visar upp leverabler till kunden och eventuella acceptanstest genomförs. Dessa möten sammanföll som planerat runt varje sprintskifte.

3.7 Backlog

För att strukturera arbetet användes *Trello*. *Trello* är en gratis programvara där kort kan sättas upp liknande post-it-lappar på en vägg. På *Trello* sattes ett kort upp för varje uppgift i projektet, och sorterades i kategorierna vad som ska göras, vad som görs, vad som görs om tid finns, vad som ska granskas och vad som har gjorts. Med hjälp av *Trello* kunde gruppen få en överskådlig bild över projektets status, och även på ett tydligt sätt se projektets framfart.

3.8 Rutiner

Vid genomförande av detta projekt avsågs vissa rutiner för att se till att arbetet skulle gå så friktionsfritt som möjligt. Utöver kodgranskning genomgicks även rutiner för dokumentation samt versionshantering.

3.8.1 Granskning

Då alla delar av projektet har legat som kort i projekthanteringssystemet *Trello* var det lätt att se vilka kort som behövde granskas då dessa markerades som redo för granskning. Gruppen specificerade i projektplanen att kod som skrivs ska granskas innan den ses som helt färdig. Granskning av kod har gjorts några få gånger formellt där hela gruppen tillsammans sett över koden och alltså de kort markerade som redo för granskning. Gruppen har sedan sett till att koden har bra struktur samt är väldokumenterad. Kodgranskningen har inte skett med något speciellt schema utan är något som gruppen avsatt tid till vid behov.

Utöver detta satt delar av gruppen stundvis tillsammans och parprogrammerade. Dels för att lättare lösa problem, dels för att koden som skrevs blev tydligare och mer lättläst.

3.8.2 Versionshantering

Då något nytt implementerades i produkten som ansågs vara fungerande lades den nya koden upp på *Git*. På detta sätt fanns relevant kod lättillgängligt för alla gruppmedlemmar. De rutiner som togs i *Git*-avseende var bland annat att göra många små *commits* för att lätt kunna återgå till tidigare fungerande versioner om någonting plötsligt skulle gå sönder. Detta skedde kontinuerligt under arbetets gång.

För att lättare hålla en tydlig struktur under arbetets gång skapades tre *repositories*, ett för serverkod, ett för klientkod och det sista för *SGCT*. Detta underlättade då gruppen kunde testa olika versioner av serverkoden mot olika versioner av klientkoden på samma dator. Vidare har varje person i gruppen hållit sig till en egen *branch* och nya uppdateringar eller tillägg av funktioner har skett på nya *branchar* som sedan slagits ihop med allas *huvudbranch*, *masterbranchen*. *Masterbranchen* sattes som en typ av back-up där den senaste fungerande versionen alltid fanns.

3.8.3 Dokumentation

När nya funktioner skrevs tog gruppen för vana att alltid infoga kommentarer för att senare, vid behov, kunna generera dokumentation. För att generera dokumentationen användes programmet Doxygen som läser igenom källfilerna till systemet och utifrån dessa tidigare nämnda kommentarer genererar en *HTML*-dokumentation som kan visas i godtycklig webbläsare.

Kapitel 4

Systemarkitektur

Tidigt i projektet bestämdes en systemarkitektur. Systemarkitekturen innefattar hur spelet ska byggas, med vilka program och kodbibliotek som ska användas. Eftersom domteatern drivs av ramverket *SimpleGraphicsClusterToolkit*[8], hädanefter *SGCT*, som beskrivs närmare i kapitel 6.3.3, blev också gruppen låst till användningen av detta ramverk. Den föregående projektgruppen hade använt 3D-biblioteket *OpenSceneGraph*[9], hädanefter *OSG*, som beskrivs närmare i kapitel 6.3.4, och det ansågs lämpligt att följa samma struktur och använda *OSG* även här. Till skillnad från den föregående gruppen, som använt Bullet som stöd för fysik i sitt projekt, valde gruppen att inte använda någon färdig fysikmotor, utan att själva skriva en vid behov. Gruppen valde att använda *Microsoft Visual Studio Ultimate 2013* som *IDE*[1]. För att kompilera koden användes kompilatorn *vc12*.

För att lösa nätverksdelen, det vill säga att externa enheter ska koppla upp sig till domteatern, valde gruppen att använda en extern *server*. Servernhanteraren som kom att användas var *SmartFoxServer*[10]. Detta beskrivs utförligare i kapitel 5.1.

Till de externa enheterna bestämde sig gruppen för att skapa applikationer som installeras lokalt. *Unity* valdes som program för att skapa dessa applikationer på grund av att det stöds av *SmartFoxServer*, samt att det tillåter grafisk implementation på ett relativt enkelt sett. Senare i projektet valdes dock *Unity* bort på grund av uppkomna problem. Detta beslut behandlas senare i denna rapport.

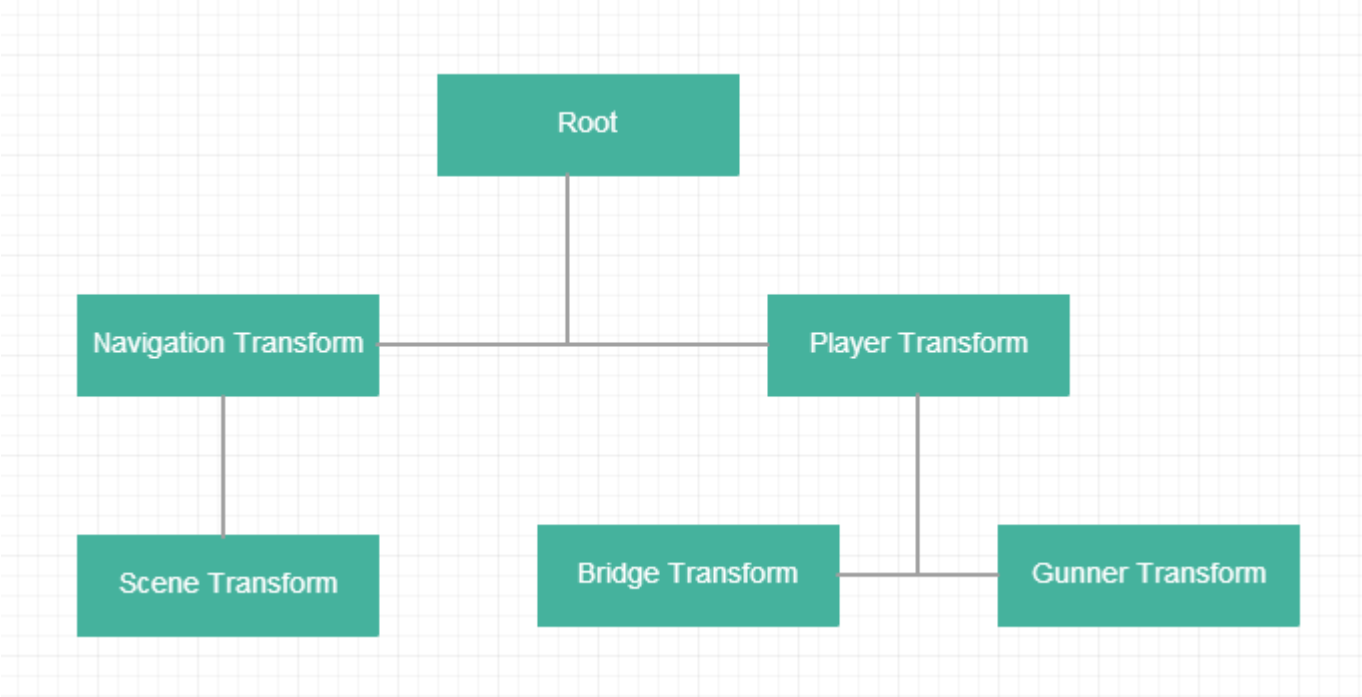
4.1 Systemmodeller

Tidigt under projektet bildades olika klassdiagram och aktivitetsdiagram. Diagrammen har hjälpt avsevärt under projektets utveckling, den tydligaste fördelen var att alla i gruppen fick samma uppfattning om hur de olika modulerna skulle se ut. Det har också hjälpt vid den individuella utvecklingen då programmeraren kunde ha diagrammet som referens då koden skrevs.

4.1.1 Scengraf

En scengraf skapades för att skapa en uppfattning på hur systemet skulle struktureras upp, en bild på scengrafen syns i figur 4.1.

[1] IDE - Integrated development environment



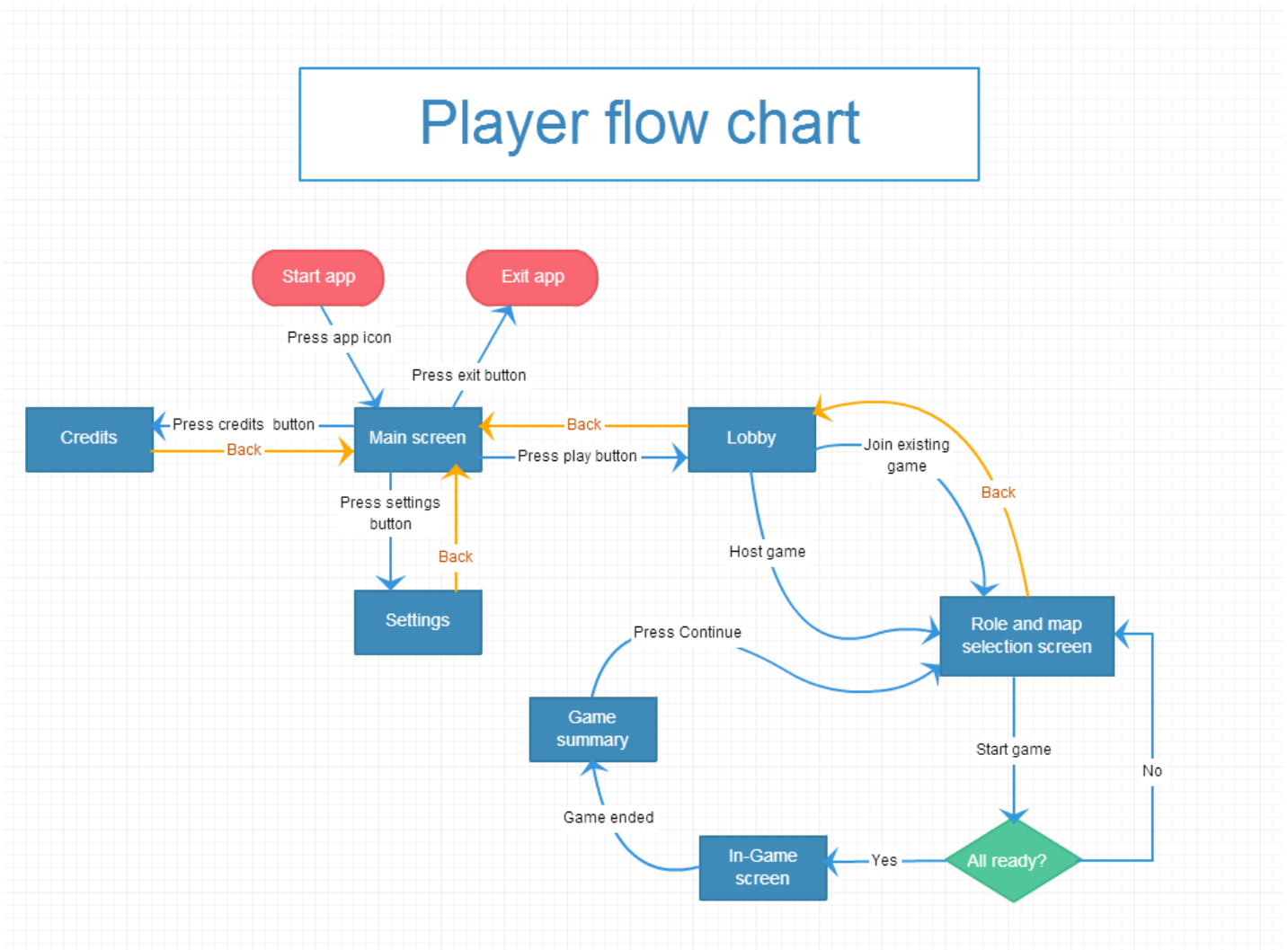
Figur 4.1: Diagram över scengrafen som använts vid renderingen. Player Transform är statisk medan Navigation Transform ändras vid input från piloten

4.1.2 Klassdiagram

Klassdiagrammet skapades tidigt under projektutvecklingen. Det skapades av anledningen att få en tydlig struktur på alla de klasser, privata funktioner och variabler som skulle behöva implementeras. Klassdiagrammet finns i bilaga A.1.

4.1.3 Flödesschema

Ett flödesschema skapades för att få en uppfattning om hur spelets menyer skulle hanteras för användaren. Flödesschemat syns i figur 4.2.



Figur 4.2: Flödesschema för användare i spelet

Kapitel 5

Nätverk

För att få mobila enheter att samverka med *SGCT* krävdes någon form av nätverksbrygga. De mobila enheterna är begränsade till vissa programmeringsspråk och utvecklingsplattformer, likaså är *SGCT* skrivet i *C++* och anslutningarna däremellan var inte helt triviala. Ett alternativ hade varit att ansluta direkt till domklienten, men då det innefattade nätverksprogrammering på en högre nivå än vad gruppen kände sig bekväm med söktes andra möjligheter. Lösningen blev att innefatta en extern *server* som agerar brygga mellan mobila enheter och domteaterns programvara. Ett schema över hur *servern* interagerar med klienterna visas i figur 5.1.

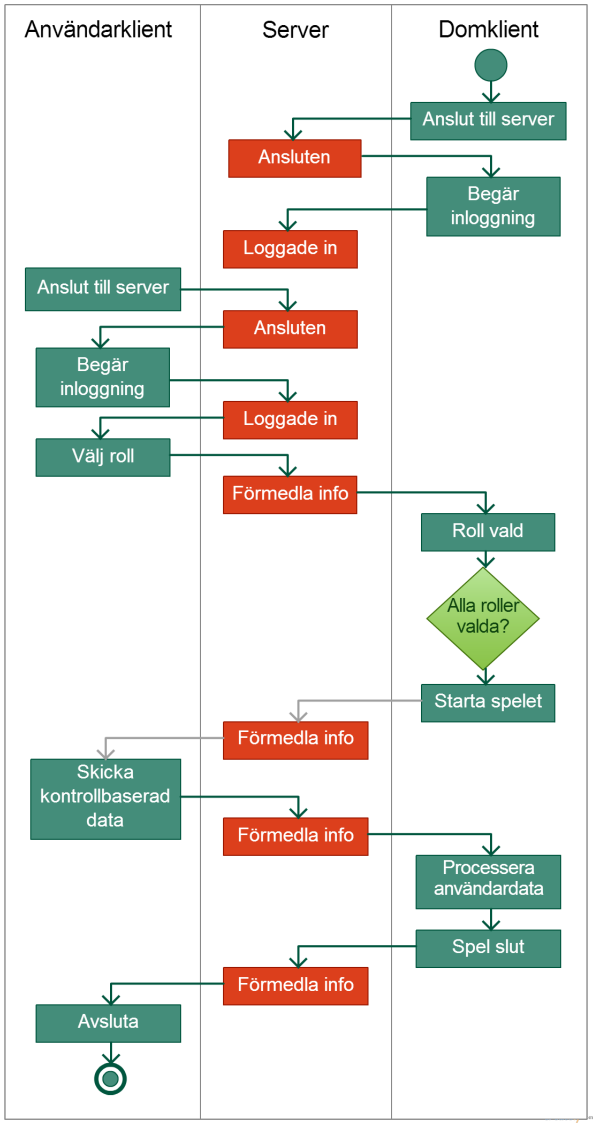
5.1 SmartFoxServer

Nätverksdelen för detta projekt hanterades främst av ett multiplattform- och tredjeparts-*SDK* kallat SmartFoxServer[10] utgiven av *gotoAndPlay()* , ett italienskt företag grundat 1999 specialiserat på spelutvecklingsteknologi [11]. SmartFoxServer sköter hantering av anslutna användare, virtuella rum, inloggning av klienter med användarnamn samt nätverksanslutningar via *websockets* och därmed *TCP/UDP*. De klientdelar som skrevs för projektet nyttjade SmartFoxServers olika *API*-er. För att underlätta vid anslutning till *server* skapades en konfigurationsfil i *XML* för den anslutande *klienten*.

5.2 API för C++

SmartFoxServers *klient-API* för *C++* släpptes 2012 och tillämpades i kombination med *SGCT* och *OSG* för att ta fram den del av systemet som körs i domteatern. Denna del fokuserade främst på att ta emot indata från de mobilbaserade klienterna. Även då SmartFoxServer stödjer *UDP*-anslutningar skickades datan via *TCP* då det ansågs tillräckligt välpresterande med avseende på det potentiella antalet anslutna klienter.

Detta *API* implementerades i en egen klass kallad NetworkManager och separerades med avseende att hålla isär olika delar av klientkoden för spelvyn. För att använda nätverksdelen instantieras ett objekt av typen NetworkManager och kallas genom funktionen init(). Klassen hanterar anslutningen till *servern* samt sätter upp lyssnare för alla nödvändiga nätverkshändelser. Den hanterar även i viss mån formatering av data mottagen från *servern* för att sedan kunna användas av *SGCT*.



Figur 5.1: Serveraktivitet

5.3 API för HTML5

För de mobila enheterna implementerades SmartFoxServers *klient-API* för *HTML5*, nyttjandes *java-script*. Mobilapplikationen skickar förfrågningar om att ansluta, logga in, välja roll samt skicka data för spelvyn. Datan skickas som snabbast en gång var 20:e millisekund och innefattar information om användarens åtgärder för hur skeppet och kanoner ska styras, om skott ska avlossas, om skeppet ska åka framåt eller hur energifördelningen i spelet ska påverkas.

5.4 Testning

För att se hur snabb svarstid systemet potentiellt kunde uppnå skrevs ett enkelt test som skickar 35 datapaket om 32 bytes (SmartFoxServer har med ytterligare flaggor så den faktiska storleket kan vara större) till servern, väntar på svar och skriver ut tiden det tog i millisekunder. Om inte antalet paket som skickats överstiger 35 så skickas ett nytt paket. Pseudokoden för hur detta går till finnes i algoritm 1 nedan.

Algorithm 1 Rekursiv benchmarking för nätverkshastighet

```
1:  procedure BENCHMARKING()
2:      startTime ← Starta timer
3:      Skapa paket om 32 bytes
4:      Skicka paket till server
5:  procedure ONSERVERRESPONSE()
6:      if numPackets++ < 35 then
7:          Spara paket som variabler
8:          endTime ← Stoppa timer
9:          Skriv ut (endTime - startTime)
10:         Benchmarking()
```

5.5 API för C#

För den senare nedlagda Unity-applikationen nyttjades SmartFoxServers C#-*API*. Data som skickades från Unity-applikationen är densamma som ovan nämnts för *HTML5*.

Kapitel 6

Spellogik och implementation

Detta kapitel innefattar hur slutprodukten har byggts och beskriver den grund som spelet vilar på.

6.1 Externa enheter

En stor del av projektet gick ut på att få de externa enheterna att fungera som spelkontroller till domteatern. Idéer om hur enheterna skulle implementeras som spelkontroller har ändrats under projektets gång.

6.1.1 Vyer

Ursprungligen var idén att de olika rollerna skulle ha en rollspecifik vy på respektive extern enhet. Piloten skulle kunna se en förstapersonsvy från kommandobryggan på rymdskeppet, och skytten skulle ha en förstapersonsvy från skeppets kanon. Ingenjörens vy var tänkt att innefatta en mer överblickande bild över skeppet för att spelaren ska kunna bedöma vilka resurser som behöver tillsättas.

6.1.2 Unity

Gruppens första idé var att skapa en applikation för att hantera vyerna till externa enheter. För att åstadkomma detta användes *Unity*. *Unity* är en grafikmotor som kan användas till spelutveckling. Anledningen att *Unity* användes är att det sparar mycket grafiskt arbete då flera rollvyer var tänkta att ha 3D-representationer. Det var också lämpligt att arbeta i *Unity* på grund av att SmartfoxServer stödjer programvaran. Problem stöttes senare på, då det visade sig att det endast är betalversionen av *Unity* som tillåter portning till *android* med inkluderat bibliotek för nätverksanslutning, som för tillfället var målplattformen. Gruppen ställdes därför inför ett val: att nöja sig med datorer som externa enheter, betala för *Unity Pro*, eller att helt byta tillvägagångssätt.

6.1.3 Spelkontroller

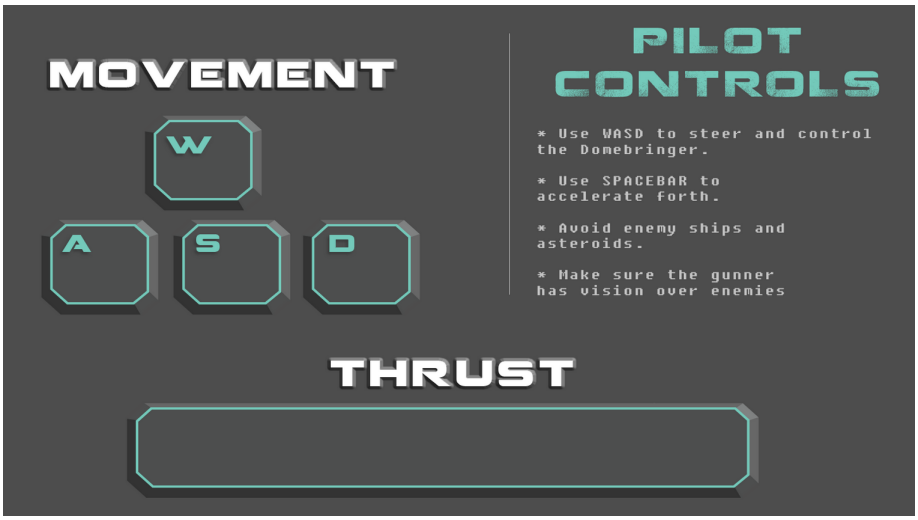
Eftersom kundens krav var att spelet ska kunna styras från externa enheter, samt att *Unity Pro* ansågs vara för dyrt, valde gruppen att byta tillvägagångssätt. Istället för att använda *Unity* utvecklades programvaran för de externa enheterna med hjälp av *HTML5*. Utöver att byta tillvägagångssätt bestämdes också en ny typ av vy att visa på de externa enheterna. De tredimensionella vyerna för rollerna valdes helt bort, och idén blev istället att endast ha ett enkelt 2D-*GUI* med rollens kontroller på enheterna.

Navigering för skytt och pilot samt kraftfördelning för ingenjörsrollen skrevs i ett canvas-element, specifikt för *HTML5*. Användarens interaktion med den mobila enheten utgår från vidrörelse av skärmen skriptat med *javascript*, vilket sedan omformaterar och skickar vidare datan till *servern* för att sedan visas på domteatern.

6.1.4 Grafiskt användargränssnitt

Målet med det nya *GUI*:t var att hålla det så enkelt som möjligt. Det var också önskvärt att spelaren skulle kunna använda kontrollerna utan att behöva kolla på dem då spelaren var tänkt att hålla blicken på domteatern under spelets gång. För att åstadkomma detta minimerades antalet knappar per roll, och knapparnas storlek gjordes så att de ska vara lätta att trycka ner utan att titta på sin externa enhet.

Eftersom användaren ska kunna spela både på mobila enheter och datorer skapades olika vyer, anpassade efter enheten. Till mobila enheter fick spelvyn visuella knappar som kan hanteras med *touch-input*, och för datorer visas istället en bild som beskriver vilka tangenter som används för att styra, se figur 6.1.



Figur 6.1: Användargränssnitt för större skärmar, exempelvis till en stationär eller bärbar dator.

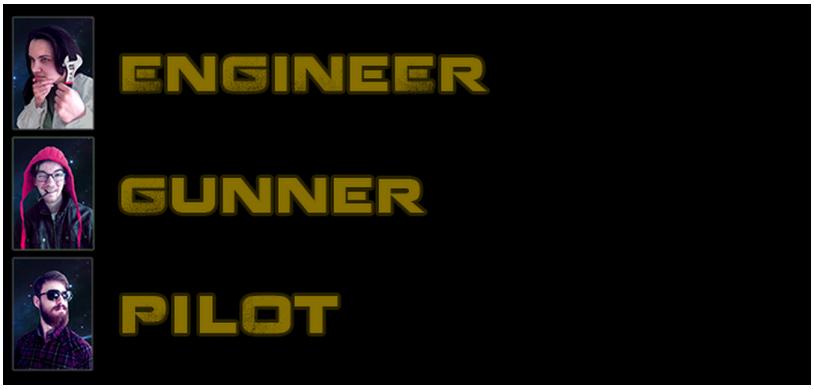
Menyer

För att hantera händelser som sker innan själva spelet startar har några menyskärmar skapats. Då webbapplikationen startas ser användaren först den meny som visas i figur 6.2, där användaren kan välja sitt spelnamn. Efter att ha valt namn kommer användaren vidare till skärmen som visas i figur 6.3, där spelrollen kan väljas. När alla roller valts startas spelet.

6.2 Teknisk beskrivning av roller

Varje roll besitter ett rollspecifikt *GUI* som agerar som rollens kontroll. Med hjälp av detta användargränssnitt kan spelaren styra den del av spelvyn som innefattas i rollens funktion. Varje knapp i dessa *GUI*:n skickar unik data till *servern*, och *servern* förmedlar sedan detta till domklienten, som använder datan för att uppdatera scenen. Nedan beskrivs rollernas *GUI*:n och datautskick mer specifikt.

Figur 6.2: Inloggnings- och välkomstvy.



Figur 6.3: Vy för att välja roll.

6.2.1 Pilot

Pilotens huvudsakliga roll är att styra det fiktiva rymdskeppet i domteatern. Detta görs via ett användargränssnitt för mobila enheter. Piloten samverkar med övriga roller genom att styra skeppet i rätt riktning för att undvika inkommande asteroider samt se till att skytten har vinkel nog att beskjuta eventuella fiender.

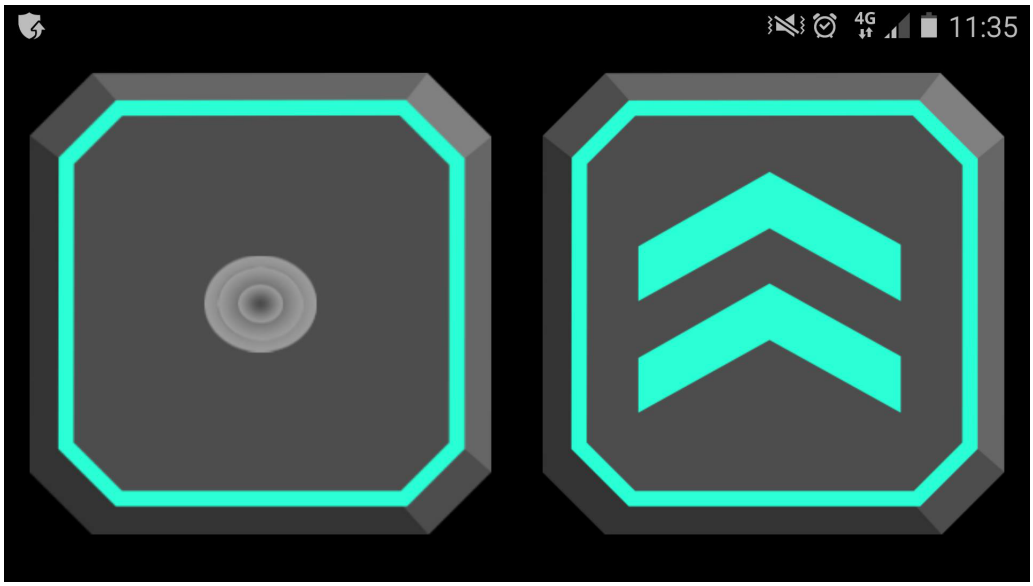
I figur 6.4 visas skyttens användargränssnitt. Pilotens *GUI* består av en styrspak för styrning samt en knapp för att accelerera. Styrspaken skickar två värden till *servern*, ett för dess position i x-led och ett för dess position i y-led. Dessa utnyttjas sedan för att bestämma vinkelaccelerationen för skeppet.

6.2.2 Skytt

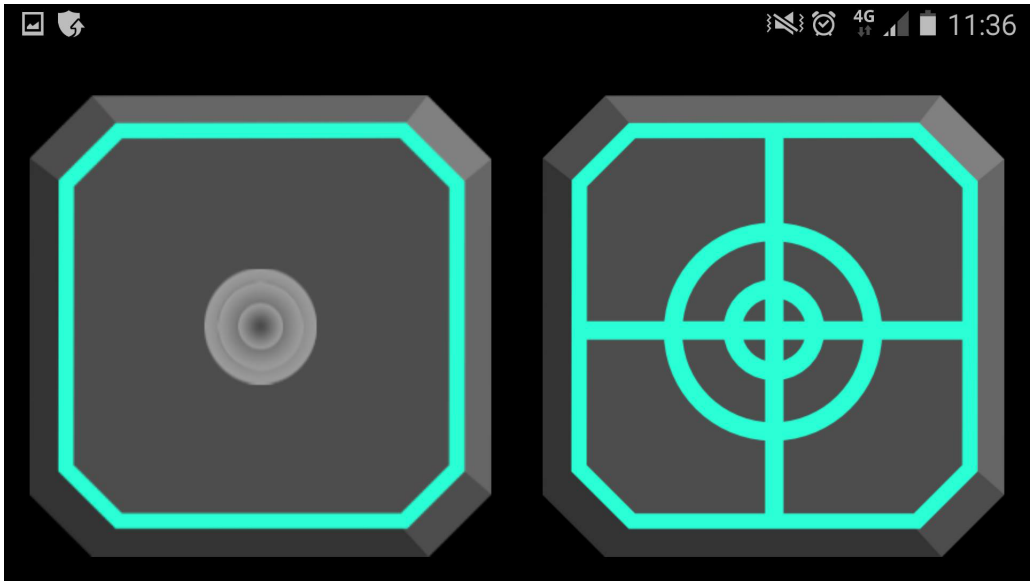
Skyttens roll går ut på att förstöra inkommande fiender genom skeppets laserkanoner. Skytten ska kunna kontrollera ett sikte samt avfyra skott i siktets riktning. I figur 6.5 visas skyttens användargränssnitt. Skyttens *GUI* liknar pilotens, med en styrspak som fungerar på samma sätt, men styr siktet istället för skeppet. Istället för en accelerationsknapp har skytten en knapp som är avsedd för att avfyra projektiler. Så länge denna knapp är intryckt talar den om för *servern* att *klienten* vill avfyra skott.

6.2.3 Ingenjör

För att skeppets olika delar ska fungera behövs ingenjören. Denna roll beslutar hur kraftfördelningen på skeppet ska skötas. Tre reglage hittas i ingenjörens arsenal. Ingenjören kan välja om kraften ska



Figur 6.4: Pilotens användargränssnitt för mobila enheter

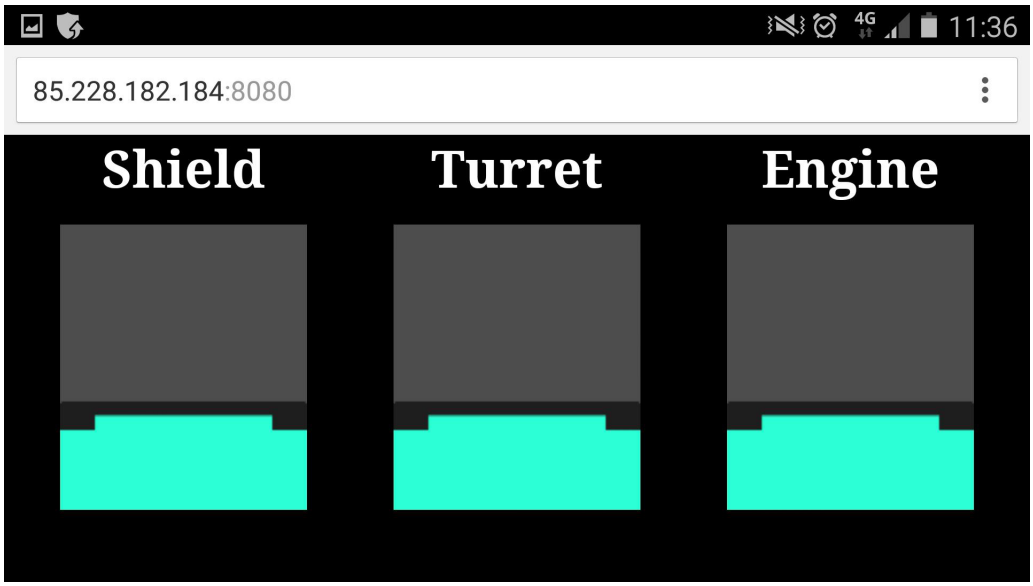


Figur 6.5: Skyttens användargränssnitt för mobila enheter

läggas på att stärka skyttens vapen, att öka motorkraften för att piloten ska kunna navigera snabbare, eller att öka sköldkraften för att ta mindre skada av inkommande föremål och fiender. De tre reglagen är beroende av varandra, vilket medför att ingenjören måste bestämma vad som är viktigt för stunden. Ingenjörens *GUI* består av dessa tre reglage som styrs med *touch-input*, och skickar olika värden beroende på hur de är inställda. I figur 6.6 visas ingenjörens användargränssnitt.

6.3 Domteaterns spelvy

Här beskrivs utvecklingsmiljön som användes till domteatern. En beskrivning av de bibliotek som använts kommer ges här. Projektet utvecklades i Microsoft Visual Studio 13.



Figur 6.6: Ingenjörens användargränssnitt för mobila enheter

6.3.1 Teori bakom en sfärisk bildyta

Domteatern består av en sfärisk yta på vilken bilden projiceras. För att belysa ytan används sex stycken projektorer, var och en driven av en egen dator. Alla datorer som driver projektorerna är del av ett datorkluster, där en av datorerna agerar *master* och de övriga agerar *slaves*. För att bilden ska projiceras korrekt behöver all information synkroniseras över klustret. Projektgruppen valde att nyttja *master-datorn* för spellogik och sedermera synkronisera händelser så som att objekt skapas eller förstörs, flyttas eller roteras.

6.3.2 Åksjuketest

Tidigt under projektets gång gjordes ett åksjuktetest i domteatern. Utifrån doktorand Alexander Bocks expertis kan eventuellt illamående undvikas genom att ha långsam navigation, samt att undvika att rotera kring vektorn parallell med riktningsvektorn. Testet genomfördes med en godtycklig *skybox* där gruppen fick känna efter hur olika rotationer i domteatern påverkade deras balanssinne. Genom tester hittades en rimlig acceptansnivå för att minimera risken för åksjuka. Denna acceptansnivå är individuell men den valda nivån i slutprodukten fanns acceptabel för medlemmarna i gruppen.

6.3.3 SGCT

Projektet utvecklades till stor del i *SGCT*, som är ett *C++* bibliotek utvecklat på Linköpings Universitet [8]. Anledningen till detta var främst för att kunna synkronisera data mellan de sex projektorerna som skapar vyn i själva domteatern. Till *SGCT* är det även möjligt att läsa in en *XML*-fil som innehåller data för hur vyn ska projiceras på skärmen, vilket underlättar arbetet då skärmen är en domteater. *SGCT* använder OpenGL för att rendera grafik.

6.3.4 OSG

OSG är ett komplett *C++* bibliotek för att skapa scengrafer samt utföra matematiska operationer på matriser och vektorer [9]. Det har även inbyggt stöd för att läsa in 3D-modeller från fil. *OSG* användes

i detta projekt för att bilda scenen som *SGCT* sedan visar på skärmen. Värt att nämna är att *OSG* är den enda scengrafen som fungerar tillsammans med *SGCT*. Enbart *OSG* har möjlighet att lämna över fönsterhanteringen, det vill säga renderingen av grafik, åt en extern process (i detta fall *SGCT*), vilket *SGCT* kräver.

6.4 Spelmekanik

Spelet som utvecklades använder en kamera som kan röra sig i tre dimensioner. Detta stycke kommer bland annat beskriva hur detta infördes i projektet. Projektiler samt kollisionshantering kommer också redogöras i detta kapitel.

6.4.1 Navigation

Då kameran måste vara låst för att projiceringen till domteatern ska fungera så anpassades scengrafen så att världen flyttas då piloten styr skeppet, istället för tvärtom. Tekniskt sett betyder det att navigationsmatrisen i scengrafen måste inverteras medan spelarens matris hålls statisk. Tidigt under projektets gång utvecklades en kamera som använde sfäriska koordinater för rotation. En tydlig nackdel med det var dock att spelaren får en tydlig upp- och ned riktning samt får koniska rörelser vid rotation längs med X- och Y-axeln då Z-axelns rotation är skild från noll. Då detta inte är optimalt för ett flygspel bestämdes det att styrningen istället skulle ske med hjälp av *kvaternioner*, vilket gav ett bättre resultat.

Kvaternioner fungerar på det sättet att matrisen i fråga roteras med en vinkel runt en vald rotationsaxel. I navigationsmatrisen utförs detta genom att de lokala X- och Z-axlarna hämtas och roteras med den nuvarande rotationshastigheten. Ekvation 6.1 visar på hur en kvaternion Q roteras med en rotationskvaternion R. conj(R) betyder i det här fallet konjugatet av R. Q används senare för att rotera själva matrisen.

$$Q = R * Q * conj(R) \tag{6.1}$$

OSG har en färdig funktion för denna typ av rotation där programmeraren inte behöver skriva rotationen i denna form.

För att ge inlevelse till spelet används krafter vid styrning samt acceleration. Detta implementeras genom att låta skeppets hastighet accelerera vid *input* från spelaren istället för att positionen ändras direkt. Enligt [13, 378] tas nästa position fram via integration via Euler-steg genom ekvation 6.2 och 6.3 nedan:

$$v(n+1) = v(n) + a(n) * dt \tag{6.2}$$

$$x(n+1) = x(n) + v(n) * dt \tag{6.3}$$

Där dt är tiden mellan två bildrutor.

Att använda accelerationer för att ge skeppet framfart gav också ett bättre resultat när styrningen skedde över ett nätverk. På grund av att *TCP* används kan paketen komma fram med ojämna mellanrum, vilket leder till att en statisk rotation som appliceras vid varje serverskick kan resultera i att rotationen ser hackig ut. Då en acceleration appliceras är rotationen mjuk även om paketen kommer fram i ojämna intervall, vilket var önskvärt.

6.4.2 Projektiler

Spelets skytt har egna kontroller som styrs genom en mobil enhet. Dessa kontroller används för att styra ett sikte som syns på skärmen. Då skytten väljer att skjuta skapas en laser som rör sig i den riktning som siktet är inställt på. För att hålla minnesförbrukningen låg så har dessa lasrar en livslängd på 3 sekunder varefter de tas bort från scenen. Detta löses genom att projektiler läggs i en lista skapad av ett av *C++* standard bibliotek.

6.4.3 Kollisionshantering

Eftersom spelet inte använder sig av någon fysikmotor bestämdes det att kollisionshanteringen skulle ske på ett så enkelt sätt som möjligt. Varje objekt får en kollisionsradie tilldelad sig när de skapas, vilken används för att jämföra om två objekt ligger innanför varandra. Nackdelen med denna typ av kollisionshantering är att utseendet på alla objekt behöver anpassas så att de blir relativt runda, så att spelaren inte märker att kollisionshanteringen inte är exakt. Detta har framförallt begränsat designen av rymdskeppsmodeller då många designer använder avlånga modeller till dessa.

6.5 Ljud

De ljudeffekter och den musik som används i spelet är skapade av projektgruppen eller, som är fallet med laserljudet, används med tillstånd från skaparen. De ljud som används i spelet är för explosioner och skott. Det skrevs också bakgrundsmusik till meny och pågående spel. För att underlätta hantering av ljud i domteatern skrevs en ljudmodul baserad på *OpenAL*, *SoundManager.cpp*. Denna hanterar laddning av ljud i formatet *WAVE*, uppspelning av ljudeffekter, uppspelning och looping av bakgrundsmusik samt möjligheten att stoppa ljud pågående ljudkällor.

Kapitel 7

Grafik

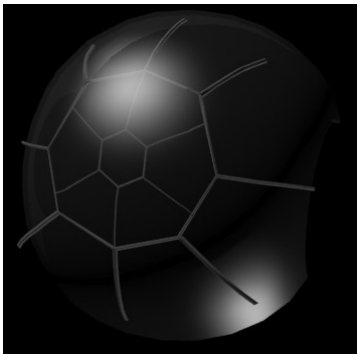
Grafiken i det här projektet har inkluderat 3D-modeller och texturer. För 3D-modellerna i spelet har *3DSMAX* använts. Någon större mängd modeller har inte krävts i det här projektet. Modeller finns för insidan av kommandobryggan, asteroider, fiendeskepp och för laserskott.

7.1 Problem vid exportering

Ett problem uppstod när exportering av modeller skulle ske i *3DSMAX*. Problemet bestod av att filformatet *.obj* och dess tillhörande *.mtl* fil gjorde modellen helt transparent. Det berodde på att materialfilen hade ett värde som stod för dess *alphakanal*. *Alphakanalens* värde var satt till noll och problemet löstes genom att sätta värdet till ett. Nämnvärt är också att endast *3DSMAX* standardmaterial kunde användas till 3D-modellerna. Gruppen testade att använda *3DSMAX* egna färdiga material men det kunde inte exporteras till *.obj* för bruk.

7.2 Kommandobrygga

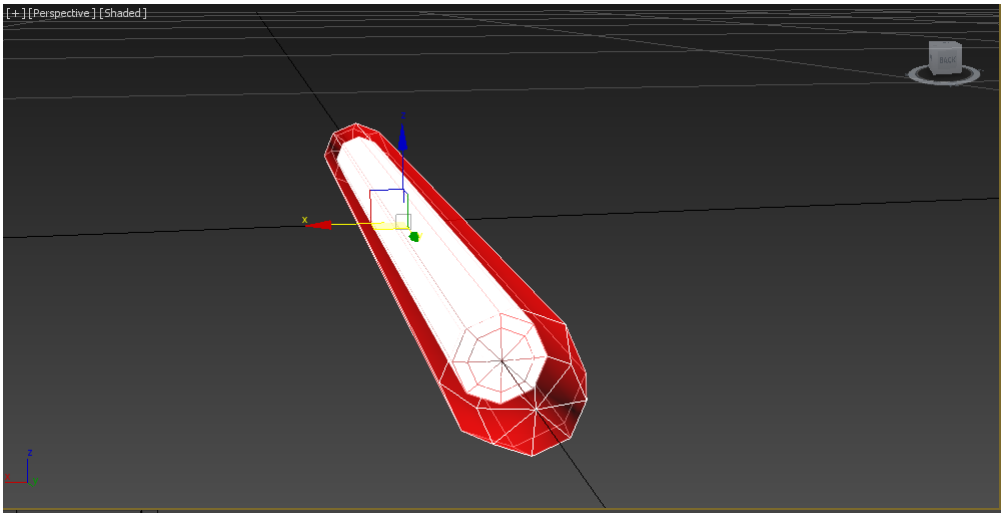
Det diskuterades om kommandobryggan skulle göras som en 2D-bild, likt den tidigare projektgruppen [1, s.14]. Det finns dock fördelar med att använda en 3D-modell, bland annat reagerar en modell på ljussättningen i scenen, där en tvådimesionell bild hade varit statisk. Att det finns andra 3D objekt i scenen gör det även svårare att ge en tvådimensionell bild illusionen av tre dimensioner. Kommandobryggan skapades först som en halvsfär, som kan ses i figur 7.1. Gruppen testade även att göra bryggan platt, men detta skulle endast se bra ut på en vanlig platt skärm. I en dom krävdes en korrekt böjd brygga för att det skulle se rätt ut.



Figur 7.1: 3D-modell för den halvsfäriska kommandobryggan till spelet

7.3 Skott

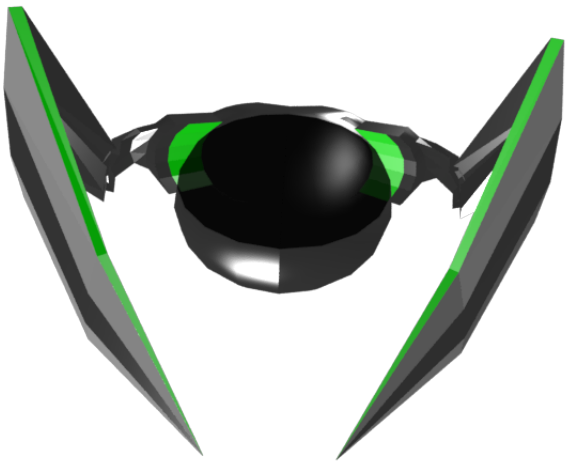
Skotten är menade att lysa inifrån. De har givits en sådan effekt med en underliggande vit cylinder som omges av an halvgenomskinlig röd cylinder. Bild på skottet ses i figur 7.2.



Figur 7.2: Skottmodell som används i spelet

7.4 Fiendeskepp

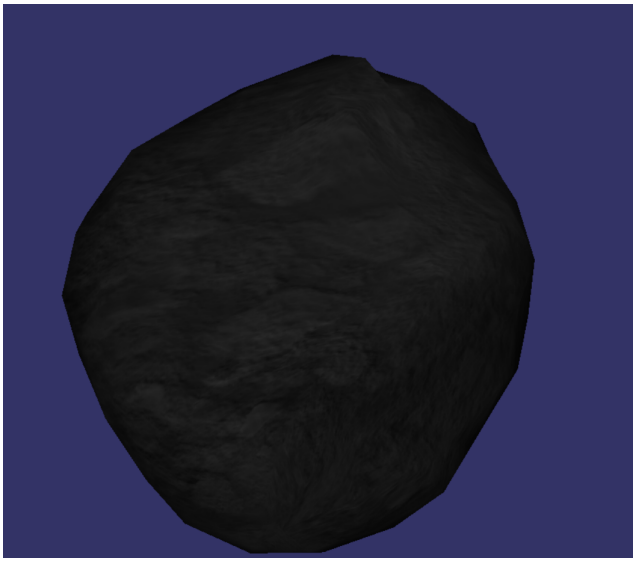
Fiendeskeppet är fritt modellerat efter ett antal konceptbilder. Det är menat att ha en rund form som tillåter användningen av en avgränsningssfär vid kollisionsdetektering. En modell av fiendeskeppet ses i figur 7.3. Färgen på fiendeskeppet uppdaterades från svart till att även vara grön och vit, på grund av att scenen i domen är så pass mörk att det blev svårt att se originalskeppen.



Figur 7.3: En prototyp av fiendeskeppet

7.5 Asteroid

Asteroiden är en sfär som manuellt deformerat och med applicerad textur. Tidigt användes den asteroid som användes i det tidigare projektet. Denna asteroid var långt mer detaljerad än vad som var nödvändigt. En av asteroiderna som gjordes för detta projekt visas i figur 7.4.



Figur 7.4: En asteroid

7.5.1 Texturer

De tvådimensionella bilder som skapats till projektet inkluderar knappar till det grafiska gränssnittet och profilbilder för rollerna. Knappar har ritats med objektgrafikprogrammet *Inkscape*. Profilbilderna är foton på projektgruppsmedlemmar som behandlats i bildbehandlingsprogrammet *GIMP*.

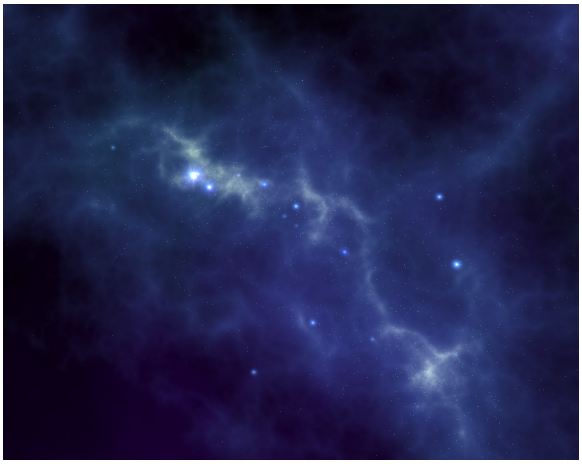
7.6 Power-ups

Till spelet skapades fyra olika *power-ups*. Dessa *power-ups* har chans att skapas då man skjuter sönder ett fiendeskepp. En bild på en *power-up* visas i figur 7.5.

Figur 7.5: En power-up som reparerar skeppet då spelaren åker på den.

7.7 Skybox

Spelets *skybox* har skapats i *SpaceScape* [12], ett gratis program för att rendera *rymd-skyboxar*. En liten del av *skyboxen* som används visas i figur 7.6.



Figur 7.6: En del av skyboxen som skapades med SpaceScape

7.8 Billboards

Till spelet har ett antal *billboards* använts. *Billboards* är ett 2D-plan som alltid är riktat mot en specifik punkt, den specifika punkten i det här spelet var kameran. Bland annat togs ett sikte för skytten fram samt att det skulle kunna användas för livsmätare och explosioner då asteroider skjuts sönder.

Kapitel 8

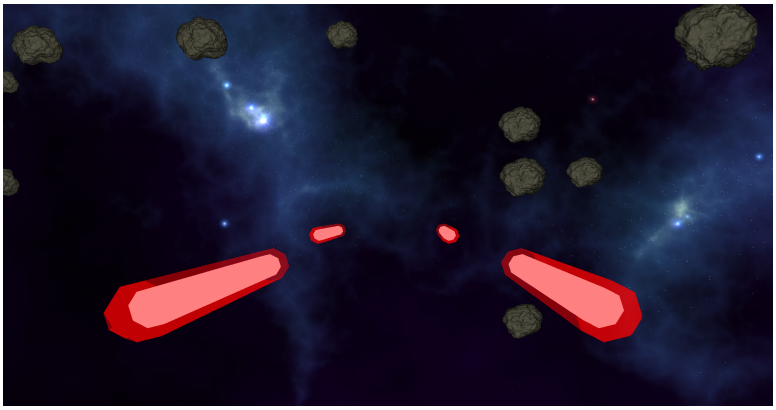
Resultat

Utifrån kundkrav, förarbete samt faktiskt arbete sammanställdes det slutgiltiga systemet. Utöver själva spelet skapades även en *Unity-klient* som av budgetskäl valdes bort. Alla samverkande delar av systemet fungerar och utgör tillsammans helheten av spelet.

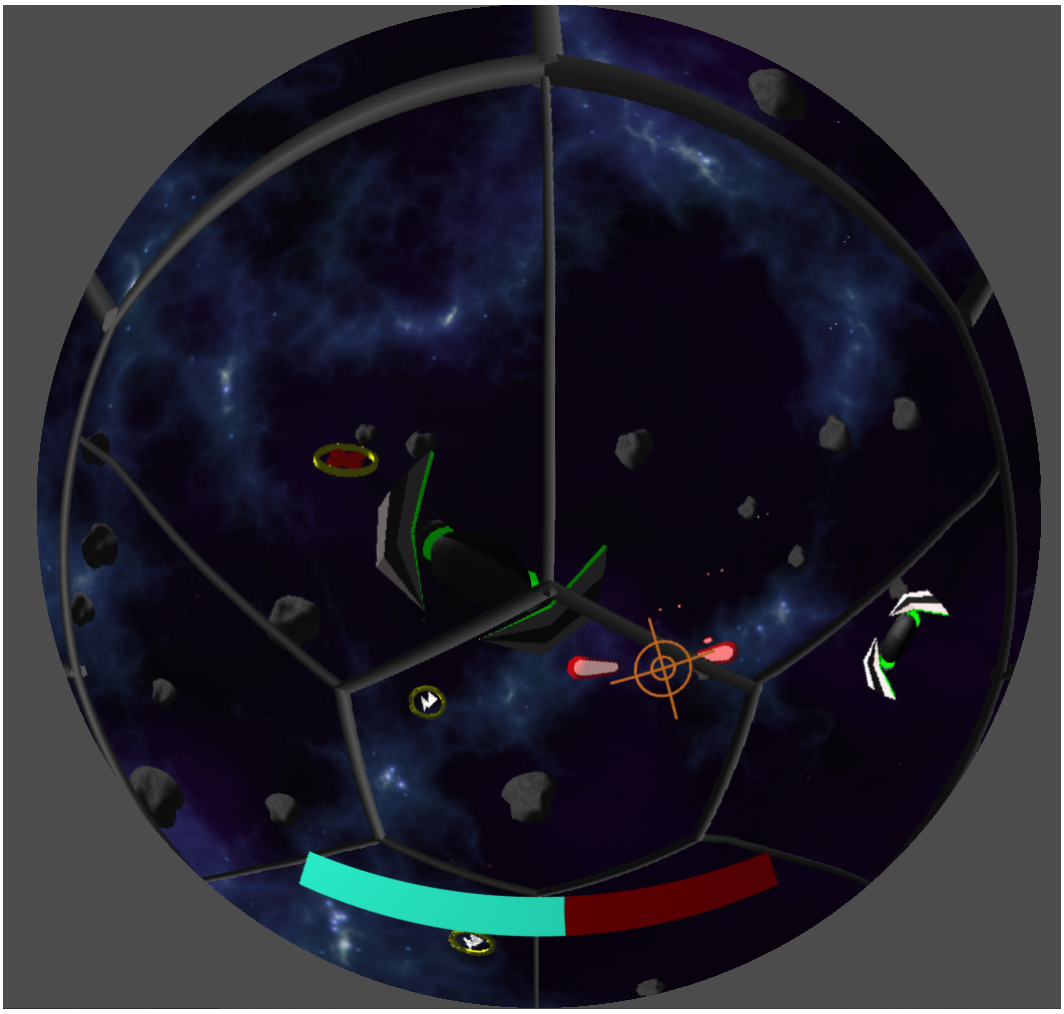
8.1 Slutprodukt

Slutprodukten är ett spelbart *multiplayerspel* bestående av en spelvy, en extern *server* samt ett grafiskt användargränssnitt för att ansluta klienter. Spelarna ansluter via godtycklig webbläsare med stöd för *HTML5* och *Websockets* oavsett enhet. Det grafiska användargränssnittet erbjuder användaren att välja ett användarnamn och i de fall namnet inte följer konventionerna specificerade av klienten antas ett slumpat användarnamn. Näst efter följer möjlighet att välja roll och spelet sätts igång när alla roller är fyllda.

Spelvyn kan köras på en *Windowsdriven* dator och visar spelvärlden inifrån en kommandobrygga, se figur 8.1 och 8.2. Då det är skrivet i *SGCT* kan produkten köras i domteatern på Visualiseringscenter C i Norrköping. Ett sikte syns som styrs av skytten vilket även avgör den riktning eventuella avlossade skott färdas mot. I denna vy syns även eventuella fiender och objekt möjliga att interagera med. Piloten styr framfart och riktning för skeppet medan ingenjören distribuerar hur skeppets kraft ska fördelas. Rollerna har olika *GUI* som visas på användarens mobila enhet.



Figur 8.1: Domteaterns spelvy projicerad på vanlig platt skärm



Figur 8.2: Domteaterns spelvy projicerad efter ”fisheye” konfiguration. Det vill säga att skärmen projicerar en vy med 165 graders vinkel.

8.2 Prestanda

Innehållet i den resulterande spelscenen är uppbyggd på ett sådant sätt att prestandan är acceptabel till en normalpresterande dator. Gruppen var till en början tveksam ifall tillämpandet av SmartfoxServer skulle funka och samtidigt få en bra responstid. Det visade sig senare inte vara något problem, utan *input* från kontroller märks direkt på huvudvyn, alltså är responstiden låg och i god mån acceptabel.

8.3 Resultat från nätverkstester

Svarstiden för systemet testades i två scenarion, ett där *server* och *klient* befann sig på samma nätverk och ett där *server* låg på en global *IP* där *klienten* satt på trådbundet hemmanätverk. Svarstiden för paket om minst 32 *bytes* låg i det första fallet mellan 9 och 14 millisekunder. När den globala *servern* användes varierade svarstiden mellan 10 och 20 millisekunder.

8.4 Unity

Applikationen skapad i *Unity*, skriven i C# och fungerar precis som webbläsarklienten. Nätverksdelen fungerar liksom spelmekaniken med skillnaden att *Unity-klienten* erbjuder en 3D-vy. Då en prisbelagd version av motorn krävs för att få med anslutningsmöjligheter vid portning till mobila enheter lades denna version på is. Även om den fungerar så uppfyller den inte kravspecifikationen då den inom satta avgränsningar går ut över budget för att fungera som ursprungligen tänkt.

Kapitel 9

Analys och diskussion

Projektgruppen är, i stort, mycket nöjd med resultatet av arbetet. Att skapa ett spel till en domteater är ett projekt som man inte får möjlighet att göra många gånger i sin karriär. Flera lärdomar har givits gruppen under arbetets gång. Dessa följer nedan.

9.1 Problem under utvecklingen

Under början av projektet skulle utvecklingsmiljön sättas upp. Gruppen valde att arbeta i *Windows* då domteaterns datorer använder det som operativsystem. Då alla i gruppen inte hade bärbara datorer med *Windows* som operativsystem blev detta ett problem. Själva utvecklingsmiljön var också problematisk att skapa. Det var många bibliotek som skulle arbeta tillsammans och alla var inte anpassade för att fungera tillsammans. Detta ledde till att många bibliotek behövde kompileras om med de inställningar som projektet krävde. Till en början installerades *SmartFoxServer* med ett tillhörande *C++ API* samt ett projekt i *Visual Studio* som kodstommen skulle skrivas i. Under utvecklingen upptäcktes det att *SmartFoxServer* enbart kördes med en 32-bitars version medan projektet var byggt som 64-bitars. Detta ledde till att *OSG* samt själva projektet i sig behövde byggas om som 32-bitarsversioner för att kunna kommunicera med *SmartFoxServers API för C++*. Ett annat problem var att *OSG* till en början inte kunde läsa in bilder som texturer. Detta berodde på att ett bibliotek som *OSG* använder, *Zlib*, saknades och behövde därmed byggas och installeras även det.

Eftersom *Unity* inte kunde användas som gränssnitt till mobilkontroller fick en annan lösning tas fram. Gruppen valde då att utveckla en *HTML5* baserad version då SmartFoxServer även har ett *API* för det. Nackdelen då blev istället att 3D-stöd för mobila enheter valdes bort eftersom det ansågs bli för mycket arbete.

9.2 Framgångar

Under projektets senare period utfördes arbetet mer centrerat. Alla gruppmedlemmar arbetade på samma plats vilket höjde arbetsprestationen hos alla gruppmedlemmar. Detta berodde bland annat på att alla höll konstant kommunikation, vilket sänkte den dödtid som kan uppstå efter att en viss uppgift har utförts.

Kundkontakten har skett effektivt genom hela projektetsgång. Alexander Bock som har stått som projektets kund har under utvecklingen även hjälpt gruppen genom att komma med idéer och tips. Ur gruppens egna perspektiv kändes det bra att han inte heller haft en alltför strikt kravspecifikation. Kravet var egentligen bara: gör ett kul rymdspel till domenöch resten lämnades upp till gruppen att

komma på.

Trots att det uppstod problem då *Unity-applikationen* lades ned så anser gruppen ändå att den lösning som valdes istället har fungerat riktigt bra. Det tog förhållandevis kort tid att utveckla en enkel sida med kontroller i *HTML5* och eftersom det kunde kommunicera med *SmartFoxServer* som redan hade satts upp så var en fungerande version klar redan dagen efter.

9.3 Framtida utveckling

Då det anses att projektet har lagt en stabil grund för ett generellt rymdspel, skulle detta projekt kunna fortsätta utvecklas i många olika riktningar. Det som är mest kritiskt är ingenjörsrollen, vilken i dagsläget är relativt trivial. Ett exempel på vad som kan utvecklas där är en radar, som visar den närliggande omgivningen i alla riktningar. Detta skulle leda till att, medan de andra spelarna enbart ser vad som är framför skeppet, kan ingenjören även se vad som är bakom det, och skulle på så sätt kunna avfärda många oförmånliga händelser. Skytten skulle också kunna få flera olika vapen, olika typer av fiender skulle kunna skapas, mer ljud till spelet med mera.

9.4 Nätverkstest

Eftersom kunden inte gett några konkreta krav på hur snabbt data skulle skickas, utan bara gett som krav att det inte ska vara någon märkbar fördröjning på data som skickas, fick gruppen bilda sin egen uppfattning. Efter att tester utförts så ansågs det att den fördröjning som uppstod låg inom godkända gränser för det krav som satts och således upplevdes som realtid. Resultatet av dessa tester beskrivs utförligt i sektion 8.3.

Kapitel 10

Slutsatser

Under projektets gång har projektgruppen kommit fram till ett antal slutsatser. Slutsatserna innefattade att de vetenskapliga frågeställningarna kunde besvaras.

10.1 Agilt arbete

Arbetet under projektets gång har skett agilt och det visade sig vara ett effektivt sätt att utveckla spelet på. Eftersom projektet utvecklades kontinuerligt var det viktigt att kunna göra ändringar under projektetsgång. Gruppen använde *Scrum* som utvecklingsmetodik. *Scrum* medförde att mycket arbete skedde under bestämda *sprints* där mål sattes upp i förväg. Utöver det användes *Trello* kontinuerligt, uppgifterna i checklistorna försökte hållas små och alla i gruppen hade mål för dagen.

10.2 Svar på vetenskapliga frågeställningar

När man spelar ett rymdspel i en domteater finns det risk för att man blir åksjuk. Hur ska produkten utvecklas för att undvika detta problem?

För att undvika problem med åksjuka används en relativt långsam navigation som gruppen genom testning fann acceptabel i den mån att balanssinnet inte påverkas på ett negativt sätt.

I det här projektet tas kod över från ett tidigare projekt. Kommer tidigare kod kunna användas på ett effektivt sätt, eller är det mer hållbart att skriva kod från grunden?

Eftersom instruktioner om hur projektet skulle kompileras saknades, var det mer effektivt i detta fall att skriva koden från grunden.

Vilka tillvägagångssätt kan antas för att lösa nätverksdelen för ett så pass omfattande projekt?

Gruppen hade två möjligheter för att lösa nätverksdelen. Den ena var att låta *klienterna* direkt koppla upp sig mot *SGCT*-klienten, det vill säga domteatern, via *Websockets* och därmed låta den klienten agera *server*. Det andra alternativet var att använda en *tredjepartsserver*. Som tidigare nämnt innefattar direktuppkopplingen till domen mer grundläggande serverprogrammering, medan *SmartFoxServer* redan har mycket färdig funktionalitet som att hantera uppkoppling och inloggning. Det andra alternativet ansågs mer önskvärt i detta projekt, då det krävde mindre arbete.

10.3 Uppfyllnad av mål

Målen som sattes i början av projektet uppnåddes under projektets gång. Spelet har fungerande spellogik och nätverkskommunikation samt frågeställningarna har blivit besvarade.

Litteraturförteckning

[1] Niklas Andersson, Daniel Camarda, Johan Eliasson, Ellen Häger, Jesper Hägerstrand, Fredrik Johnson och Oscar Westberg, 2014, *TNM094 - Mediatekniskt Kandidatprojekt Game of Domes*, Linköpings Universitet.

[2] Gunnar Gunnarsson, *TCP/IP-handboken*, Pagina AB, 1998.

[3] Jan Skansholm, *Java direkt, med Swing*, Studentlitteratur AB, 2010.

[4] Musegames, 2012, *Guns of Icarus*, [ONLINE] Tillgänglig på: http://gunsoficarus.com/. [Datum då sidan besöktes: 23 April 2015]

[5] Thom Robertson, 2013, *Artemis | Spaceship Bridge Simulator*, [ONLINE] Tillgänglig på: http://www.artemis.eochu.com/. [Datum då sidan besöktes: 23 April 2015]

[6] Shari Lawrence Pfleeger och Joanne M. Atlee, *Software Engineering, Fourth Edition, International Edition*, Pearson 2010

[7] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmnith, Andrew Hunt, Ron Jeffires, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaberm Jeff Sutherland, Dave Thomas, 2001 *The Agile Manifesto*, [ONLINE] Tillgänglig på: http://www.agilealliance.org/the-alliance/the-agile-manifesto/. [Datum då sidan besöktes: 19 februari, 2015].

[8] Linköpings Universitet, 2014, *SGCT - Simple Graphics Cluster Toolkit*, [ONLINE] Tillgänglig på: https://c-student.itn.liu.se/wiki/develop:sgct:sgct. [Datum då sidan besöktes: 22 April 2015].

[9] OpenSceneGraph, 2014, *OpenSceneGraph*, [ONLINE] Tillgänglig på: http://www.openscenegraph.org/. [Datum då sidan besöktes: 24 April 2015]

[10] gotoAndPlay(), 2004, *SmartFoxServer - Platform overview*, [ONLINE] Tillgänglig på: http://smartfoxserver.com/overview/platform. [Datum då sidan besöktes: 22 April 2015].

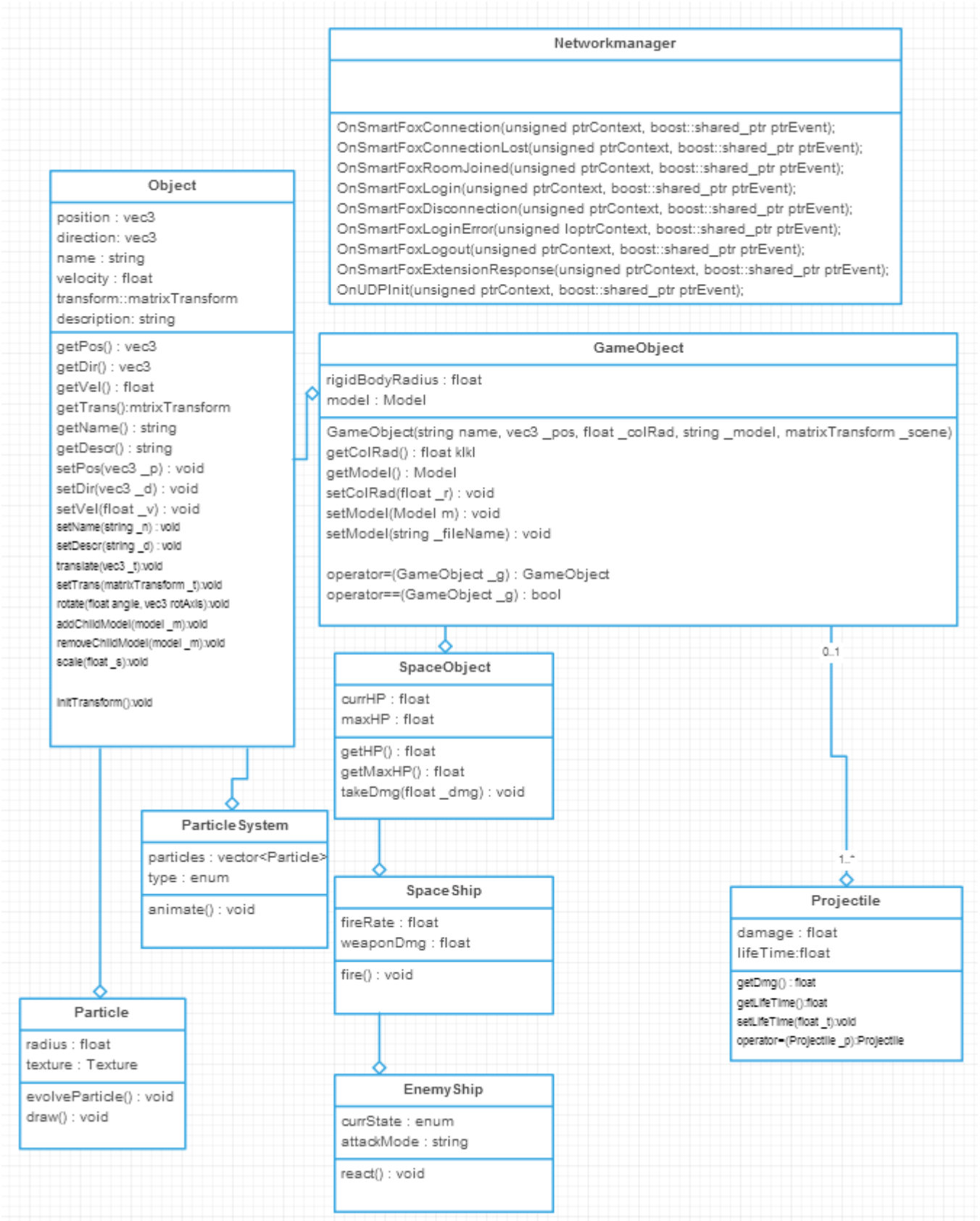
[11] gotoAndPlay(), 2004, *SmartFoxServer - Company*, [ONLINE] Tillgänglig på: http://smartfoxserver.com/company. [Datum då sidan besöktes: 22 April 2015].

[12] Alex Peterson, 2010, *Alex Peterson - Spacescape*, [ONLINE] Tillgänglig på: http://alexcpeterson.com/spacescape/ [Datum då sidan besöktes: 22 April 2015]

[13] Lennart Ljung, Torkel Glad, *Modellbygge och simulering, andra upplagan*, Studentlitteratur AB, 2004.

Bilaga A

UML-diagram



Figur A.1: Diagram över de klasser som implementerats. Networkmanager hanterar SmartFoxServer kommunikationen

Bilaga B

Bidragande personer

- **Henrietta Rydfalk: Grafik**: Skapade modeller och texturer för användargränssnitt samt assisterade Torsten med kod för *SGCT* och *OSG*.
- **Lars Bergman: Scrum master**: Ansvarade för *backlogstrukturen*, skapade modeller, implementerade *serverkod*.
- **Max Jonsson: Nätverk**: Skrev grunden för webbklienten i *HTML5*, skrev ljudhanteringen i *OpenAL* samt komponerade ljud och musik, ansvarade för och implementerade *serverkod* samt skrev nätverksdelarna av spelvyklienten och webbklienten.
- **Tobias Pihl: Unity**: Skapade *Unityapplikationen*, implementerade *serverkod* samt satt med webbklienten i *HTML5*.
- **Torsten Gustafsson: *SGCT*/OSG**: Ansvarade för- och skrev koden för *SGCT* och *OSG*. Förde mötesprotokoll.
- **Karl Johan Lundin Palmerius: Examinator**: Hjälpte till att lösa tekniska och matematiska problem med *SGCT*.
- **Alexander Bock: Kund**: Hjälpte till att lösa tekniska problem och kom med värdefulla idéer angående struktur och spelmekanik.
- **Miroslav Andel: Teknisk chef, LiU** Bidrog med kunskap och tips om ombyggnad samt omkompilering av externa tredjepartsbibliotek samt delade med sig av sin kunskap om *TCP* och *UDP*.
- **Niklas Andersson: student** Niklas var med i föregående projektgrupp och hjälpte till att komma igång med projektet.
- **Ellen Häger: student** Ellen var med i föregående projektgrupp och hjälpte till att komma igång med projektet.