Objektorienterad 3D-spelsutveckling i jMonkeyEngine med iterativ arbetsmetodik

Jonas von Essen Julia Ortheden Fredrik Viberg Philip Nord

LSP310 Kommunikation och ingenjörskunskap
Chalmers tekniska högskola
29 Maj 2017

Sammanfattning

Objekt i ett ramverksbaserat tredimensionellt utrymme innehåller ofta data såsom position, storlek och färg, men även datamodellen behöver åtkomst till dessa egenskaper. Det kan alltså bli svårt att separera modell och vy utan att skapa överflödiga objekt eller duplicerad data. I detta projekt, genom vilket ett Java-utvecklat 3D-spel för PC har tagits fram, tacklades denna svårighet. Arbetet skedde efter agil modell över sex iterationer och bit för bit växte en fungerande MVC-modell fram. Under projektets gång erhölls många viktiga erfarenheter kring hur god objektorientering uppnås i praktiken och hur man bäst planerar en effektiv utvecklingsprocess. Resultatet blev ett spel kallat Not Enough Space, där spelaren styr ett UFO vars mål är att kidnappa så många kor som möjligt från jordens yta på två minuter. Kidnappningen försvåras av exploderande satelliter och en fullständigt rasande bonde som gör allt i sin makt för att rädda sitt boskap. Det ursprungliga problemet med att i koden separera modell från vy gäckade gruppen under större delen av projektet men fick till slut en i hög grad tillfredsställande lösning.

Innehåll

1 Inledning

Spelutveckling kan ofta misstas för en oseriös bransch då estetiska specialister utgör större delen av branschens ansikte utåt. Bakom dessa finns dock en bred yrkeskompetens inom både datorteknik och mjukvara och faktum är att spelindustrin är en av de absolut största krafterna för utveckling av digital teknik [1].

Ett digitalt spel består av mjukvara som exekveras och de filer som bildar spelets tillgångar, exempelvis musik, bilder och 3D-modeller. Spelets mjukvara kan innehålla tusentals eller miljontals rader kod. En stor del av denna kod är till för att hantera grundläggande saker som grafik, fysik, användarinput och uppspelning av ljud. Dessa aspekter, på den tekniska nivån, har inte mycket med själva spelet att göra. Därför återanvänder man ofta befintliga bibliotek, grafikmotorer eller spelramverk. Dessa är skrivna av professionella programmerare för att underlätta utvecklingen av småskaliga såväl som större spel.

Att utveckla bra programvara kräver struktur, både i kod och arbetssätt. Ju bättre design som planerats i förväg, desto lättare blir själva programmeringen. Det blir även lättare att återanvända, uppdatera, förbättra och underhålla koden. I boken *Object-Oriented Design Using Java* förespråkar Dale Skrien att kod ska vara elegant, vilket innebär att den ska uppfylla ett flertal krav för att läsning och fortsatt arbete med koden ska vara så lätt som möjligt [2]. Ett enkelt sätt att åstadkomma detta enligt honom är att använda sig av så kallade designmönster, det vill säga fungerande och väl beprövade lösningar till vanliga problem.

”Don’t call us, we’ll call you.” är ett citat som beskriver hur de flesta ramverk fungerar [3]. Det innebär att ramverket har den övergripande kontrollen över applikationen, och anropar det utökande programmet på olika sätt. Utöver de tidigare nämnda aspekterna så kan ett spelramverk därmed även hantera delar av spelets logik och data, eller underlätta programmeringen av sådan.

Detta är i regel användbart, men kan leda till svårigheter när det gäller att skriva modulär kod. Något som Skrien nämner som en av grundpelarna inom objektorienterad programmering och som även är viktigt för utvecklingsgrupper som använder ett iterativt arbetssätt.

1.1 Syfte

Rapporten syftar till att redogöra för och utvärdera framtagningen av 3D-spelet Not Enough Space, ett PC-spel för en spelare utvecklat i Java. Den analyserar och diskuterar några av de problem som uppstår när objektorienterad spelutveckling kombineras med 3D-spelsramverk. Vidare evaluerar den effekten av ett iterativt arbetssätt i relation till resultatet.

2 Teori

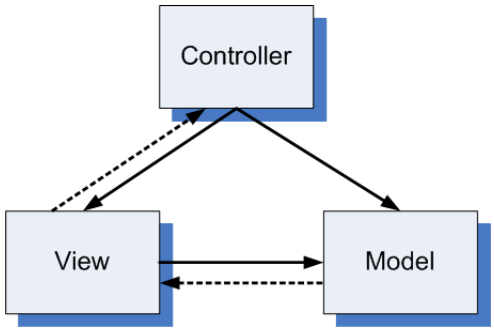
Teorin är uppdelad i två distinkta delar. Den första består av teknisk information om objektorienterad programvaruutveckling och grundläggande teori om 3D-spelsramverk. Andra delen beskriver vad det innebär att jobba agilt med ett projekt.

2.1 Objektorienterad programvaruutveckling

I objektorienterad programvaruutveckling, OOP, är varje program uppdelat i objekt som interagerar med varandra [2]. Ett objekt utgörs av en samling kod med relaterade tillstånd och beteenden och skapas utefter klassmallar [4]. OOP handlar främst om att designa och strukturera de olika klasser och beroenden som behövs, på ett sätt som förenklar fortsatt arbete och håller en klar modularitet [2]. Optimalt ska programstrukturen fungera likt en svart låda, det som finns där inne ser inte ut och det som ligger utanför kommer inte åt det där inne. På så sätt behålls kontroll över hur objekten beror på varandra. Detta möjliggör utbyte av klasser eller paket utan omstrukturering av koden, samt återanvändning av dessa delar även i andra program.

2.1.1 Designmetoden Model-View-Controller

Det mest övergripande designmönstret, som är viktigt att tänka på redan i de första stegen av skapandeprocessen, är Model-View-Controller (MVC). MVC innebär att man ska separera koden i tre delar: datamodellen, vyn och kontrollern (Figur 1) [5]. Denna separation innebär att man senare kan byta ut dessa delar utan att de andra slutar fungera, exempelvis kan man uppdatera eller byta ut sin grafikimplementation utan att behöva refaktorera datamodellen.



Figur 1: *Demonstration av hur MVC-strukturen kommunicerar. De heldragna pilarna betyder att paketen har en association medan de sträckade pilarna betyder oberoende kommunikation.*

Modellen är den del som innehåller programmets data och sköter alla beräkningar, samt ansvarar för uppdateringar av den tillgängliga informationen. För korrekt upprätthållning av MVC ska modellen vara oberoende av både vyn och kontrollern.

Vyn kan med tillgång till modellen visa upp information för användaren visuellt på skärmen eller via ljud. Vyn bestämmer exakt hur modellens data ska visas upp [5].

Kontrollern fungerar som ett tunt lager som sköter en stor del av kommunikationen mellan vyn och modellen. När användaren interagerar med programmet, exempelvis via knapptryck eller menyval konverterar kontrollern informationen till kommandon som skickas till modellen. Beroende på innehållet kan kontrollern även välja en ny vy att presentera för användaren [5].

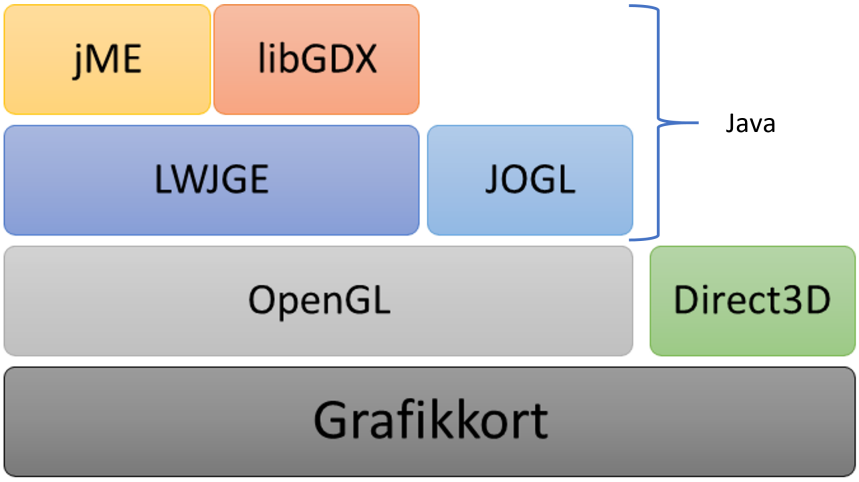
2.2 3D-spelsramverk för Java

Det ramverk som används vid utvecklingen av ett spel kallas för en spelmotor [6]. Metaforen passar bra då motorn är den viktigaste komponenten i en maskin. Den är kraftfull och samma motor kan användas i allt från bilar till lyftkranar.

På samma sätt kan en spelmotor exempelvis nyttjas för att utveckla pusselspel såväl som äventyrsspel, även om den är specialiserad för en specifik typ. För mindre projekt utnyttjas oftast befintliga spelmotorer för att bespara resurser, medan stora spelföretag brukar utveckla egna spelmotorer för att kunna få full kontroll över motorns flexibilitet och prestanda.

Inom mjukvaruutveckling arbetar programmeraren med att skapa kod i ett visst språk som, när koden kompilerats, kan exekveras på en viss plattform. Programmeringsspråk, ramverk och bibliotek bildar tillsammans den miljö som utvecklaren arbetar i. Utvecklingsmiljön är anpassad efter de resurser och den omfattning som utvecklingen innefattar. En stor variabel för programspråk är abstraktionsnivån. Abstraktion låter utvecklaren programmera på en högre nivå utan att behöva tänka på detaljer, även om koden inte går att optimera lika mycket som om den hade skrivits i ett språk med låg abstraktionsnivå.

Java är ett programmeringsspråk med hög abstraktionsnivå, vilket leder till att det är intuitivt att utveckla i men dessvärre inte kan exekveras direkt av grafikkortet. För att underlätta åtkomsten till grafikkortet finns det ett antal API:er (Application Programming Interface) som bygger på varandra. OpenGL är det API som grafikkortet kan exekvera. Ovanför det finns flera bibliotek som erbjuder Java-utvecklare åtkomst till OpenGL-anropen. Dessa bibliotek är fortfarande primitiva, vilket har lett till vidareutveckling i fullfjädrade spelmotorer som exempelvis jME och libGDX (Figur 2) [7].



Figur 2: *Bibliotek för 3D-programmering på olika abstraktionsnivåer. Längst ner sitter grafikkortet, som utför den faktiska utritningen. Ovanför det syns flera nivåer av API:er, som alla krävs för att översätta från Java till grafikkortet. De som är markerade med Java är språkspecifika.*

Dessa två kodexempel beskriver liknande saker på olika abstraktionsnivå (Figur 3). Det vänstra kodstycket målar en fyrkant på skärmen, en gång. Det högra stycket skapar en tredimensionell kub som lagras i ett utrymme och sedan målas automatiskt under varje bildskärm, i rätt perspektiv och med rätt färg och storlek. Det som kopplar samman dessa stycken är att det vänstra stycket anropas under varje bildskärm, någonstans djupt inne i ramverket, som följd av att programmeraren har skrivit det högra. Ramverket skapar alltså en väldigt hög abstraktionsnivå, och automatiserar det underliggande arbetet.

OpenGL (C++)	**jMonkeyEngine (Java)**

glBegin(GL_POLYGON);
 	
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(0.5, 0.0, 0.0);
glVertex3f(0.5, 0.5, 0.0);
glVertex3f(0.0, 0.5, 0.0);
 	
glEnd();
	Box box1 = new Box(1,1,1);

Figur 3: *Olika abstraktionsnivå. Kodstycket till vänster är väldigt maskinnära och åstadkommer ganska lite, medan det högra anropar en lång rad underliggande kod.*

2.2.1 Ramverket jMonkeyEngine

JMonkeyEngine är en spelmotor med öppen källkod som utvecklas ideellt av ett team på åtta personer [8]. Det är en typisk modern 2D- och 3D-spelsmotor och inkluderar bland annat stöd för inläsning och rendering av 3D-modeller, fysikberäkningar för dynamiska objekt, unifierad plattformsoberoende indatahantering och ljuduppspelning i 3D för musik och ljudeffekter.

2.3 Agila arbetsmetoder

Att arbeta agilt innebär att man följer ett antal riktlinjer och principer som är tänkta att göra arbetet mer effektivt, flexibelt och hållbart. Dessa övergripande principer publicerades år 2001 av en grupp programmerare i det som kallas ”Manifestet för agil systemutveckling” [9]. De fyra principerna följer nedan.

2.3.1 Individer och interaktioner framför processer och verktyg.

Här eftersöks flexibilitet genom att låta individernas interaktioner sinsemellan styra inriktningen på projektet snarare än att strikt följa särskilda procedurer för kommunikationen deltagarna emellan. Tanken är att en mer organisk och obehindrad kommunikation kommer att göra arbetet mer effektivt.

2.3.2 Fungerande programvara framför omfattande dokumentation.

Man eftersträvar att undvika överflödig och alltför tidskrävande dokumentation till förmån för snabbare utveckling av projektet. Istället för att utgå från detaljerade specifikationer börjar man med en mer övergripande vision och låter de mindre viktiga detaljerna bli ett resultat av arbetet med projektet.

2.3.3 Kundsamarbete framför kontraktsförhandling.

Förespråkare för agila arbetssätt menar att kunden oftast inte har en fullständig bild av hur denne vill ha slutprodukten innan arbetet har satt igång. Även om bilden är klar är sannolikheten stor att den ändras och att nya önskemål och idéer dyker upp under utvecklingens gång. Därför föredras ett nära samarbete med kunden under hela processen och inte bara i inledningsfasen.

2.3.4 Anpassning till förändring framför att följa en plan.

Då IT-branschen i allmänhet ses som relativt oförutsägbar är det inte säkert att en viss plan håller hela vägen från idé till färdig produkt. Agila arbetsmetoder söker att kringgå detta genom ständig utvärdering och förbättring av planen.

Detta sker tydligast genom nyttjandet av iterativa arbetssätt, vilket innebär att projektet utvecklas inkrementellt och att fungerande delleveranser av den färdiga produkten sker med jämna mellanrum. Detta gör att projektet tidigt kan komma till viss användning och gör det lättare att utvärdera vilka eventuella förändringar som bör göras [10].

3 Metod

Här beskrivs och resoneras kring de val av arbetsmetoder och utvecklingsmiljö som gjorts under projektets gång för att underlätta och effektivisera arbetsprocessen.

3.1 Utvecklingsmiljö

Till projektet Not Enough Space användes utvecklingsmiljön IntelliJ. Detta trots att jME har en specialutvecklad miljö kallad jMonkeyEngine SDK som möjliggör direkt skapande och redigerande av 3D-scener, 3D-modeller och material [11]. Den främsta anledningen till att IntelliJ valdes framför jMonkeyEngine SDK var att samtliga gruppmedlemmar tidigare hade utvecklat i IntelliJ och att det kändes som en kraftfullare utvecklingsmiljö. För att skapa de 3D-modeller som krävdes användes då i stället open source-programmet Blender.

Hantering av projektets mappstruktur och externa beroenden, främst jME, förenklades genom användningen av verktyget Apache Maven. I början uppfattades Maven komplicerat, men efter en del konfiguration kunde hela applikationens byggsteg automatiseras.

3.1.1 Versionshantering

För att möjliggöra för alla medlemmar i gruppen att arbeta med källkoden parallellt användes versionshanteringssystemet git. Git är ett distribuerat system, vilket innebär att varje gruppmedlem håller hela versionshistoriken i sitt arbete [12]. Genom att använda GitHub.com som fjärrserver kunde alla gruppmedlemmar få tillgång till den senaste versionen av källkoden och direkt publicera nya versioner på GitHub. En annan fördel med versionshantering är hur arbetsuppgifter konkretiseras och synliggörs genom att varje modifikation av källkoden motiveras med ett s.k. ”commit message”, som är en kort redogörelse för vilka ändringar som skett.

Versionshanteringen användes även för att publicera gruppmötesprotokoll samt tekniska dokument.

3.2 Iterativt arbetsätt

Utmärkande för projektets upplägg var det iterativa arbetssättet. Alltså arbetet i mindre cykler för att förbättra exempelvis olika funktioner genom prototyp, test, analys och förbättring. Detta var genomgående för hela projektet och i tillägg bestämdes det tidigt att alla projektdeltagare skulle arbeta på samtliga övergripande delar av projektet.

Inför majoriteten av iterationerna användes GitHubs funktion med "issues", det vill säga uppdrag som spaltades upp och kunde färdigmarkeras eller delegeras. De flesta iterationer som utfördes var ungefär en vecka långa och sedan utvärderades och analyserades arbetet samt eventuella problem som då uppkommit. Fokus låg på att anpassa sig till förändring framför att strikt följa den ursprungliga planen. Parallellt med iterationerna var också föreläsningar med målsättningar och fokuspunkter för att styra gruppens arbete.

4 Implementation

Implementationen delades upp i sex iterationer varav samtliga utom den första pågick i en vecka. Varje iteration bestod av ett antal väldefinierade uppdrag som fördelades mellan gruppmedlemmarna. Arbetet i respektive iteration beskrivs nedan. I iteration 1-3 låg fokus på att lära känna ramverket och i iteration 4-6 byggdes modellen om och spelets funktionalitet tog form.

4.1 Iteration 1-3: Gruppen lär känna ramverket

Den första iterationen sträckte sig över tre veckor. Till en början låg fokus på att studera ramverk och på speldesign. Målet var att fort bygga en första prototyp samt under tiden lära sig om ramverket. Kod som skrevs placerades i godtyckliga klasser utan någon överenskommen struktur för programmet. Senare uppstod en del frågetecken kring domänmodellen och dess koppling till det befintliga ramverket. Under en föreläsning påpekades vissa problem med MVC-struktur i samband med JMonkeyEngine. Gruppen insåg då att dessa problem fanns i den dåvarande programstrukturen, vilket ej uppmärksammats tidigare. Därför förflyttades en stor del av arbetsfokuset till att lösa strukturproblemen med målet att få en välfungerande objektorienterad MVC-struktur.

4.2 Iteration 4-6: En bättre modell tas fram

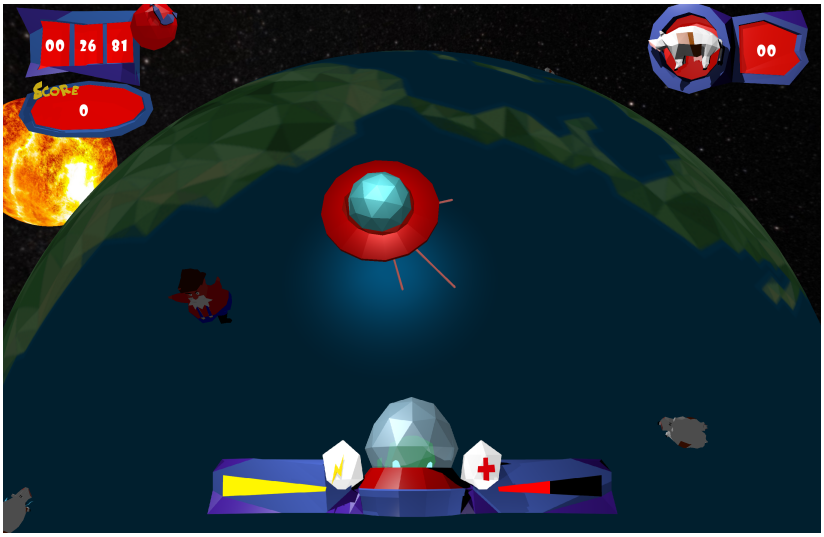
Inför den fjärde iterationen planerades en ny MVC-modell. Den här gången skulle modellen bygga på att positionsdata lagrades även i datamodellen, för att möjliggöra rörelse- och tillståndsberäkningar utanför ramverket. Utöver omstruktureringen skulle en liten mängd funktionalitet läggas till. Det upptäcktes

dessvärre att dubbellagringen och hanteringen av vektortransformationer utanför ramverket var både komplicerad och problematisk ur en objektorienterad synvinkel, så modellen tänktes om ytterligare en gång och implementerades i sin helhet mot slutet av iterationen. De resterande uppdragen sköts upp.

Planen för iteration fem var att färdigställa och utöka spelets funktionalitet, eftersom modellen ansågs komplett och elegant. Därför startades arbete med nya spelmekaniker. Den sista iterationen innehöll stora förbättringar inom spelets grafik och användargränssnitt. Den ursprungliga idén byggdes även ut med extra funktionalitet, och spelet i sin helhet färdigställdes.

5 Resultat

Det färdiga programmet är en prototyp av ett PC-baserat arkadspel för en spelare lokaliserat i jordens atmosfär. Spelaren kontrollerar ett UFO. som navigerar runt jorden och har som uppdrag att stråla upp så många kor som möjligt under två minuter. Samtidigt måste spelaren akta sig för flera hinder. Spelets källkod är strukturerad enligt designmönstret Model-View-Controller, med en datamodell helt utan kopplingar till ramverket.



Figur 4: *Skärmdump från en spelomgång. I övre vänstra hörnet syns återstående tid och totala poäng. I högra hörnet syns antal infångade kor. Längst ner på skärmen visas spelarens energi- och hälsomätare.*

5.1 Kodstruktur

Kodens struktur omarbetades vid ett flertal tillfällen och har därför sett mycket olika ut i olika skeden av utvecklingen. Slutligen uppnåddes dock ett resultat som av gruppen uppfattades som ett lyckat förverkligande av den objektorienterade programmeringens grundprinciper. Datamodellen har distinkt separerats från ramverket och en robust eventbuss har upprättats som en del i kommunikationen mellan programmets olika delar. Större delen av beräkningarna återfinns i modellen, som därför utan särskilt stora besvär skulle kunna ryckas loss från ramverket och användas i ett annat sammanhang. Kontrollagret har tunnats ut i så hög grad som möjligt så att det enbart fungerar som ett kommunikativt skikt mellan modellen och vyn, vilken håller sig passiv och enbart uppdateras från modellen. Dessutom är modellen helt testbar, vilket var ett av målen som sattes upp inför projektet. Kommunikation från modellen till den 3D-rymd den förväntas påverka sker även genom en abstraktion av alla tillhörande ramverksspecifika funktioner.

5.2 Programmets funktionalitet

Den resulterande programvaran är funktionellt sett centrerad kring spelet, vilket har ett arkad-liknande format med högt tempo och korta nivåer. En begränsad mängd kringliggande funktionalitet som pausmeny och topplista blev också implementerat.

5.2.1 En Spelomgång

En spelomgång är enligt implementationen 120 sekunder lång. Tiden räknas hela tiden ner och mängden tid kvar presenteras på skärmen (Figur 4). Användaren kan styra sitt skepp med olika tangenter på tangentbordet. Spelaren kan interagera med sin omvärld genom att stråla upp kossor som ger poäng, och bråte som tynger ner skeppet. Strålen förbrukar energi, vilket är en begränsad resurs som laddas upp när strålen inte används. Det går även att återfå energi genom att samla upp powerups", i form av små lådor som ligger i omlopp runt planeten.

Det finns flera hinder implementerade för att öka utmaningen. Förutom begränsningen av tid och energi har skeppet en hälsomätare. Denna sjunker då satelliter i omlopp kring jorden krockar med skeppet, samt då högafflar kastade av en arg bonde träffar skeppet. När spelarens hälsomätare är tom avslutas spelet direkt.

5.2.2 Topplista

Efter en avslutad spelomgång presenteras spelarens poäng i relation till andra spelare på en topplista. Topplistan är i slutprodukten endast lokalt lagrad och vyn kan nås genom att en spelrunda tar slut eller direkt från programmets huvudmeny.

6 Diskussion

Återkommande under hela projektet har varit problemet med hur man på bästa sätt kan använda sig av ramverket och samtidigt få en fristående modell. I och med valet av ramverk försvårades implementationen av det objektorienterade programmeringssättet som förväntades i kursen. Mycket tid och energi lades på att abstrahera den egna modellen från ramverkets modell. Inte förrän i iteration fem färdigställdes programmets domänmodell, trots att rekonstruktionen var planerad redan i iteration tre. I detta projekt hade både tid och energi kunnat sparas genom att en bra modell byggts från början, innan koden började implementeras. Samtidigt behärskade gruppen från början inte riktigt ramverket och det uppkom inte förrän i iteration 2 att valet av ramverk och projekt skulle kunna komma att komplicera arbetet med att hålla projektet objektorienterat efter kursens riktlinjer. Att spelet byggdes med en välstrukturerad MVC-modell ansåg dock gruppen vara väldigt viktigt eftersom det förenklar arbete med olika spellägen och spelmekaniker. Med hänsyn till detta är gruppen väldigt nöjda med slutresultatet, trots att arbetsgången inte blev så rak som planerat från början.

En annan lärdom som tagits med från projektets är att spelets kollisionsdetektor hade kunnat implementerats annorlunda. Gruppen tog beslutet att kollision skulle kontrolleras noggrant av ramverket, som sedan skulle notifiera modellen, i stället för att modellen själv skulle kontrollera kollision med hjälp av avstånd. Det senare hade varit en lättare lösning och i efterhand har det insetts att den inte är sämre för det.

6.1 Arbetssätt

Det iterativa arbetssättet, och grupparbete i allmänhet, bygger mycket på att ett visst mål kan nås genom att gruppmedlemmarna arbetar på separata delar parallellt, något gruppen gjorde och också upplevde som väldigt effektivt. Både gruppmötena och uppdragen på GitHub var till stor hjälp för att koordinera gruppens arbete. Utan dessa och god planering hade det lätt kunnat bli kaotiskt. Nu var det lätt att se vem som jobbade med vad och vad som behövde göras. Att träffas för gruppmöte två gånger i veckan och utvärdera hur arbetet gått var ett effektivt arbetssätt då problemen snabbt kunde uppmärksammas och lösas.

6.2 Användarupplevelse

Mycket fokus har lagts på kameravinklar, styrning och spelgrafik för att optimera användarupplevelsen. Värt att uppmärksamma är kamerafunktionen som låter skeppet styra fritt, utan att kameran följer med, i en cirkel i mitten och sedan återgår till normalt tredjepersonsläge när skeppet styrs utanför den zonen. En lätt accelerationseffekt gör att skeppets rörelse känns någorlunda realistisk. En annan funktion som bidrar till användarupplevelsen är de egenhändigt skapade modellerna och animationerna, som i kombination med ljudeffekterna förstärker den tecknade och lekfulla känslan i spelet.

6.3 Fortsatt utveckling

Det finns idéer om hur spelet skulle kunna vidareutvecklas genom en andra fas som börjar där det nuvarande spelet tar slut. I den andra fasen kan spelaren, efter att ha kidnappat så många kor som möjligt, lämna jorden för att bege sig till sin hemplanet genom ett farligt asteroidbälte. Detta skulle ge spelet en handling och göra det mer till en liten historia. Naturligtvis skulle spelets befintliga fas också kunna utvecklas genom att andra typer av strålbara objekt läggs till, fler fiender införs och skeppet bestyckas med någon typ av vapen. Den objektorienterade kodstrukturen som uppnåddes under projektet skulle förmodligen göra sådana påbyggnader relativt enkla att implementera, eftersom man lätt kan skapa och kontrollera nya objekt.

En teknisk förbättring som skulle kunna implementeras är att koppla loss spelets beroende av scenen. I nuläget skulle det vara väldigt enkelt att byta scen så länge denna är sfärisk, men skulle man vilja ha en platt yta så blir det problematiskt eftersom all navigation är implementerad som rotation runt planetaxeln. En annan förbättring som gruppen skulle genomföra är att göra topplistan nätverksdelad, vilket skulle ge spelet ännu en dimension då man skulle kunna mäta sig även med andra spelare.

7 Slutsats

Att göra ett objektorienterat spel är en komplex uppgift som tar mycket tid. Framför allt att kombinera det objektorienterade programmeringssättet med ett 3D-spelsramverk medför många komplikationer. För att effektivisera arbetet bör en stor del av tiden i början av projektet användas till att skapa en bra systemdesign så tidigt som möjligt, vilket underlättar både arbetsfördelning och implementation för resten av projektet. Att jobba i iterationer och låta slutprodukten vara anpassningsbar under projektets gång tycks vara ett lyckat arbetssätt.

Referenser

[1] C. Hadzinsky, "A look into the industry of video games past, present, and yet to come," CMC Senior Thesis, Claremont McKenna College, 2014. [Online]. Tillgänglig: http://scholarship.claremont.edu/cmc_theses/842/

[2] D. Skrien, *Object-Oriented Design Using Java.* New York, NY: McGraw-Hill, 2008.

[3] B. Appleton. (2000) Patterns and Software: Essential Concepts and Terminology. [Online]. Tillgänglig: http://www.sci.brooklyn.cuny.edu/~sklar/teaching/s08/cis20.2/papers/appleton-patterns-intro.pdf

[4] (2017) The Java™ Tutorials. Oracle Corporation. [Online]. Tillgänglig: https://docs.oracle.com/javase/tutorial/java/concepts/

[5] (2017) Design Patterns - MVC Pattern. Oracle Corporation. [Online]. Tillgänglig: http://www.oracle.com/technetwork/articles/javase/index-142890.html

[6] J. Gold, *Object-Oriented Game Development.* Harlow, Essex: Pearson Education Limited, 2004.

[7] (2017) LWJGL - Lightweight Java Game Library. Lwjgl.org. [Online]. Tillgänglig: https://www.lwjgl.org/

[8] R. Kusterer, *JMonkeyEngine 3.0 Beginner's Guide.* Birmingham-Mumbai, UK: Packt Publishing, 2013.

[9] K. Beck. (2001) Manifesto for Agile Software Development. [Online]. Tillgänglig: http://agilemanifesto.org/

[10] (2017) 12 Principles Behind the Agile Manifesto. Agile Alliance. [Online]. Tillgänglig: https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

[11] (2017) The jMonkeyEngine3 SDK based on netbeans. jMonkeyEngine. [Online]. Tillgänglig: https://github.com/jMonkeyEngine/sdk

[12] R. Preissel, B. Stachmann, B. Kurniawan, and C. Mayle, *Git: Distributed Version Control–Fundamentals and Workflows.* Montreal, Quebec: Brainy Software, 2014.