Planetsimulator

Johannes Olegård Anton Olsson

14 maj 2013

1 Programbeskrivning

1.1 A

Vi ska skriva ett program som simulerar ett 2D-universum. Användaren kan skapa planeter i olika storlekar, massor, färger, hastigheter, accelerationer, etc. och planeter kommer sedan interagera med varandra (t.ex. skapa omloppsbanor).

1.2 B

Vi har skrivit ett program som simulerar ett 2D-universum. Programmet kan ses som ett s.k. sandbox-spel. Användaren kan skapa planeter i olika radier, massor, färger, och hastigheter. Planeter interagerar med varandra genom gravitation (acceleration) och kollisioner.

Projektet finns på github: `https://github.com/jesajx/inda12-proj`

1.3 C

Programidén ändrades inte ändrats så mycket, men har till viss del byggts vidare på (t.ex. med kollisioner och ljud).

2 Användarbeskrivning

2.1 A

Applikationens användargrupp är seende mellan ca 13 och 90 år med åtminstone lite datorvana som är intresserade av att experimentera med fysik. Användaren förutsätts ha viss kunskap i relevanta fysiskaliska begrepp (t.ex. vet ungefär vad massa och radie är) då programmet ej förklarar dessa. Vidare förutsätts användaren förstå engelska då text i programmet är på engelska.

Programmet skulle exempelvis kunna användas av högstadiebarn för att med experiment lära sig hur gravitation, etc. påverkas av massa, hastighet, etc.

2.2 B

Applikationens användargrupp är egentligen vem som än vill experminentera med fysik. Användaren behöver ha minimala kunskapar i fysiska begrepp (främst massa och radie) och förstå engelska.

Applikationen uttnyttjar skärm, mus och tangentbord, så användaren behöver förstås kunna se och ha åtminstone en fungerande arm. Programmet spelar upp ljud vid vissa handlingar, men hörsel är inget måste.

Programmet skulle exempelvis kunna användas av högstadiebarn för att med experiment lära sig hur gravitation påverkas av massa, hastighet, etc.

Programmet är också lite av ett sandbox-spel som många skulle kunna tycka är kul att bara leka runt i.

2.3 C

Inte så mycket har ändrats över uppfattningen av användargruppen. Eventuellt att färre fysik kunskaper troligtvis krävs eftersom användarna är tänkta att experimentera sig fram.

3 Användarscenarier

3.1 A

3.1.1 Scenario 1

Användaren är en datalog på kth som går första året. Fysikkunskaperna är goda. Användaren vill simulera ett solsystem med 3 planeter i omloppsbana runt en stjärna.

Programmet startas med ett fönster. Vänstra delen av skärmen består av en meny. Längst ner i mitten av skärmen är en tickande klocka (en visare men inga siffror). Resten av skärmen är svart.

I menyn finns parametrar som t.ex. massa och radie att ställa in. Användaren skriver in stjärnans egenskaper och klickar sedan i mitten av skärmen där det då skapas en ifylld cirkel.

Användaren ställer in nästa planet och håller inne mus-knappen där planeten ska skapas och drar sedan pekaren åt det hållet initial hastighet ska vara. Medans användaren drar med musen så syns en pil från planeten till pekaren med text brevid om hur hög hastigheten blir när användaren väl släpper.

En streckad ellips visar hur omloppsbanan kommer bli. När användaren släpper så börjar den nyskapade planeten röra sig i en ellips runt stjärnan. Användaren repeterar mönstret för 2–3 planeter men märker denna gång att när hen håller på att placera (musknapp nertryckt och vektorn för hastighet synes) en ny planet så pausas simulationen för att underlätta placeringen.

3.1.2 Scenario 2

Användaren har viss erfarenhet av datorer (bl.a. spel) och fysikkunskaperna är minimala. Programmet startar likt beskrivet i Scenario 1. Även denna användare ska skapa ett solsystem med en stjärna och några planter i omloppsbana runt den.

Användaren testar att klicka ut lite planeter utan att ändra några parametrar. De rör sig inte så användaren försöker dra i en av dem. Planeterna och klockan nere i mitten pausas samtidigt, planeten markeras och en hastighetpil visas från den till muspekaren. Användaren släpper musknappen och planeten får fart - ut från skärmen. Användaren försöker få skärmen att följa efter och upptäcker att det går att pause med space-knappen och att zooma med mushjulet.

Planten ges nu lite fart tillbaka - lite lägre denna gång.

Användaren provar nu att ändra parameterarna på planeten. Ökad storlek verkar inte ge gravitation, men massa verkar göra det.

Användaren ökar massan rejält planeten och alla andra planter börjar röra sig snabbare och snabbare mot den.

Vissa planeter missar den stora planeten; då får större hastighet när de kommer nära och slungas på så sätt förbi. Andra planeter åker rakt in i planeten och så att de smälter samman. Användaren noterar att massan hos den stora (fortfarande markerade) planeten ökar i menyn till vänster.

Användaren stannar den stora planeten genom att sätta dess hastighet till noll i menyn till vänster och börjar sätta ut planeter en sträcka ifrån den - med olika hastigheter, åt olika håll och med låga massor.

Efter några försök har användaren lyckats få några planeter att åka i en elliptisk bana runt den stor planeten.

3.2 B

3.2.1 Scenario 1.1

Användaren är en datalog på kth som går första året. Fysikkunskaperna är goda. Användaren vill simulera ett solsystem med ca 3 planeter i omloppsbana runt en stjärna.

Programmet startas med ett fönster. Vänstra delen av skärmen består av en meny. Resten av skärmen är i i stort helt svart, bortsett från lite text.

I menyn finns parametrar som t.ex. massa och radie att ställa in. Användaren skriver in stjärnans egenskaper och höger-klickar sedan i mitten av skärmen där det då skapas en ifylld cirkel.

Användaren ställer in och placerar nästa planet. När användaren klickar håller hen höger musknapp och drar åt det hållet initial hastighet ska vara. Medans användaren drar med musen så syns en pil från planeten till pekaren med text brevid om hur hög hastigheten blir när användaren väl släpper.

En streckad ellips visar hur omloppsbanan kommer bli. När användaren släpper så börjar den nyskapade planeten röra sig i en ellips runt stjärnan. Användaren repeterar mönstret för 2–3 planeter men märker denna gång att när hen håller på att placera (musknapp nertryckt och vektorn för hastighet synes) en ny planet så pausas simulationen för att underlätta placeringen. Alla planeter börjar inte gå i omloppsbana som planerat, men med några försök lyckas användaren.

3.2.2 Scenario 2.1

Användaren har viss erfarenhet av datorer (t.ex. spel) och fysikkunskaperna är minimala. Programmet startar likt beskrivet i *Scenario 1.1*. Även denna användare ska skapa ett solsystem med en stjärna och några planeter i omloppsbana runt den.

Användaren testar att klicka ut lite planeter utan att ändra några parametrar. De rör sig inte så användaren försöker klicka och dra i en av dem. Ett snabbt vänsterklick på planeten markerar den och ändrar i menyn till vänster.

Drag med vänster musknapp flytter planetens position. En cirkel visar hur planeten hamnar efter flyttningen och en linje dras under dragningen mellan planetens nuvarande position och musen. Vidare visas den nya positionens koordinater bredvid cirkeln. Efter en planet flyttats börjar den däremot inte röra sig.

Drag med höger musknapp ger däremot en hastigheten i motsvarande riktning. Alla planeter pausas medans användaren drar. Från den den högerklicks-dragna planeten visas en pil till muspekaren och bredvid visas en siffra. Ju längre pilen är, ju större blir siffran - alltså måste

det vara hastigheten. Användaren släpper höger musknappen och planeten får fart - ut från skärmen. Användaren försöker få skärmen att följa efter och upptäcker att det går att pause med space-knappen, att zooma med mushjulet och att förflytta skärmen med piltangenterna.

Planeten ges nu på samma sätt lite fart tillbaka - lite lägre denna gång. Användaren måste dra flera gånger för att få planeten att åka åt rätt håll - högerklicks-drag lägger till hastighet, snarare än ställer in en hastigheten.

Användaren provar nu att markera planeten och ändra dessa parameterar i menyn till vänster. Ökad radie verkar inte öka gravitationkraften, men massa verkar göra det.

Användaren ökar planetens massan rejält och alla andra planter börjar röra sig snabbare och snabbare mot den. Användaren märker att en linje ritas från den markerade planeten i den riktning den förflyttar sig.

Vissa planeter missar den tunga planeten; de får större hastighet när de kommer nära och slungas på så sätt in bakom planeten och iväg. Andra planeter åker rakt in i planeten och med ett pling-ljud studsar mot den och flyger iväg.

Användaren stannar den stora planeten genom att i menyn sätta dess hastighet till noll. Användaren testar därefter metodiskt att sätta ut planeter på olika avstånd och hastigheter från den tunga planeten. PLaneterna sätts ut med låga massor.

Användaren märker att linjen som ritas från markerade planeter i rörelse ibland böjer sig - den visar alltså hur planeten kommer att röra sig i framtiden. Med hjälp av detta lyckas användaren efter några försök få några planeter att åka i elliptiska banor runt den tunga planeten.

3.2.3 Scenario 3

Scenariot börjar likt de två ovan. Användaren börjar däremot genom att klicka på ”templates” i vänstermenyn. Hen får då upp ett fönster med en lista på olika förbyggda universum. Höger halva av fönstret finns ett fält med en beskrivning av markerad template. Längst när på skärmen finns två knappar: ”cancel” och ”load”. Användaren väljer i listan passande system för uppgiften (enkelt solsystem) och klickar på load. På skärmen skapas ett fungerande solsystem. Användaren zoomar ut med mushjulet för att får en bättre blick av systemet.

Användern provar ladda ett nytt solsystem, dubbelstjärnigt denna gång. När användaren klickar på load tas föregående solsystem bort och ersätts med det dubbelstjärniga.

3.3 C

Inte så mycket ändrades från original-scenariorna. Främst har små ändringar i hur programmet ser ut och betéer sig lagts till.

4 Testplan

4.1 A

Vi testar programmet framförallt genom användartestning.

Användartestningen ska ske genom att ta tag i lämplig person i godtycklig kth korridor och få denne att utföra en eller flera av följande uppgifter:

1. Skapa ett solsystem med en planet och en tillhörande måne.

2. Justera omloppsbanan på en yttre planet i ett existerande solsystem med 2 orbitaler så att planeten hamnar innanför den andra.

3. Skapa ett dubbelstjärnigt solsystem eventuellt med en tillhörande planet.

4. Skapa en lite galax. Många stjärnor i mitten och fler hela solsystem som kretsar runt dessa.

De två senare uppgifterna är lite svårare och vi räknare inte med att användarna klarar dessa. Däremot vill vi gärna se hur de försöker använda programmet för att lösa problemen.

Innan användaren börjar ska vi kort förklara med ord (utan att visa) vad vi vill att hen ska försöka utföra. Vi ska också tydligt förklara att eventuella misslyckanden beror på brister i det grafiska gränssnittet och inte användarens bristande intelligens. Vi behöver tydligt anmärka att programmet är under utveckling och att det därför är det fullt möjligt att buggar gör uppgifterna omöjlig. Det bör finnas otydligheter i det grafiska gränsnittet och vi ber därför användaren att säga högt vad hen tänker så att vi kan förstå vad som behöver ändras/förtydligas för en användarvänlig design.

4.2 B

Vi testade programmet när det blivit stabilt och de flesta funktioner är fungerande genom användartestning.

Användartestningen skedde genom att ta tag i lämplig kth student efter en föreläsning och försöka få denne att utföra följande uppgifter:

1. Skapa ett solsystem med en planet och en tillhörande måne.

2. Skapa ett dubbelstjärnigt solsystem eventuellt med en tillhörande planet.

Vi tog en student från inda12 som har goda fysik kunskaper och passar in på användar scenario 1. Innan användaren började förklarade vi kort med ord (utan att visa) vad vi ville att hen skulle försöka utföra. Vi förklarade att det grafiska gränssnittet ännu inte är användarvänligt, vilket är anledningen till att vi utför testet, och att eventuella misslyckanden inte beror på användaren.

Vi anmärkte tydligt att programmet är under utveckling och att det därför är det fullt möjligt att buggar gör uppgifterna omöjliga. Vi bad användaren att säga högt vad hen tänker så att vi kan förstå vad som behöver ändras/förtydligas för en användarvänlig design.

Efter ca 1 minut så blev det uppenbart att kontrollerna inte var intuitiva nog då användaren inte förstod hur han skulle göra. Vi förklarade kort vilka mus- och tangent-knappar som gör vad och lät användaren fortsätta.

TODO för stor massa och det blev otydlig infinite bug. framtidsyn slutade fungera korrekt, visade endast nuvarande hastighet.

Vi upptäckte bl.a. att även fast användaren hade goda fysik kunskaper så var det svårt att uppskatta vilka storleksordningar på massor som är lagom för för planeter på olika avstånd och med olika hastigheter.

Det var otydligt hur färg ändrades (RGB-kod) och testpersonen tyckte förinställda färger kunda ha varit bra (”BLÅ”, ”GRÅ”, etc.).

Detta förtydligades genom att ”RRGGBB” står i fältet om användaren inte ändrat det. Färger ansågs som en mindre del av programmet och då libgdx inte tycktes ha ett färgväljarfönster inbyggt, deprioriterades vidare förbättringar av färgväljningen.

Det var otydligt om planetparametrar hade justerats efter ändringar i fält. Användaren måste klicka enter i varje fält som ändrats. Ingen respons ges om ändringen applicerats.

Det förtydligades genom att rubriktexter till fälten (t.ex. ”Mass”) skrevs i gulfärg om värdet i texten inte överensstämmde med värdet motsvarande planet hade, annars i vit färg.

När användaren höger-klicks-drar i en planet kan en hastighet adderas till planetens nuvarande hastighet. Användaren försökte bl.a. ändra riktining på eller stanna planeten med funktionen, vilket var problematiskt då det är svårt och användaren fick dra många gånger.

Detta förbättrades genom att shift + höger-klicks-drag sätter hastigheten snarare än adderar till den.

Att kunna markera flera planeter och t.ex. justera deras gemensamma hastighet hade varit användbart. Till exempel om användaren lyckas skapa ett solsystem, men att de i sin helhet rör sig bortåt, skulle användaren kunna nollställa den gemensamma hastigheten så att planeterna fortsätter kretsa runt varandra, men att systemet i sin helhet inte rör sig bortåt.

Fler-planets-markering lades till: shift + vänster-klick markerar flera planeter utan att avmarkera redan markerade planeter. Fälten justerades så att t.ex. hastighet och position var en ”medelvärde” av de markerade planeternas värden.

Det kunde vara svårt att navigera mellan planeter.

T-knappen kan användas för att slå av och på att skärmen följer den markerade planeten. N-knappen kan användas för att skärmen ska hoppa till och markera nästa planet (i ordningen de skapats).

Det var svårt att avgöra hur mycket skärmen var ut-zoomad från simulatorvärlden utan att sätta ut planeter och jämföra.

Zoom-nivån lades till i nedre högra hörnet av skärmen, tillsammans med en ”klammer” som brukar finnas på många kartor. Dessa visar hur stort avståndet är mellan taggarna på klammern är med nuvarande ut-zoomning.

Om inga värden angivits i fälten när en planet skapas, slumpas värden fram. Dessa värden var inte alltid rimliga. Till exempel hade planeterna för låga massor för att de skulle generera märkbar gravitation.

Slump-värdena justerades.

Någon form av hjälp-meny som beskriver kontrollerna hade varit användbar.

En knapp i vänstermenyn lades till, vilken öppnar ett kortfattat hjälp-fönster när den trycks.

Vidare upptäcktes flera buggar, t.ex. att programmet kraschar om ”remove”-knappen klickas när ingen planet är markerad.

4.3 C

Vid testerna märkte vi att det kanske inte var rimligt eller nödvändigt med de svårare uppgiterna. P.g.a. tidsbrist hann vi bara med en person och då programmet fortfarande inte var helt klart upptäckte vi en del buggar (dvs. inte bara design-fel).

Annars gav resultaten mycket användbar feedback, så pass att alla funktions-idéer som dök upp inte kunde implementeras i tid.

5 Programdesign

5.1 A

Programmet ska använda opengl för det grafiska med hjälp av bibloteket libgdx[1]. Artemis entity system[2] kommer användas för att lättare separera koduppgifter i mindre klasser där varje klass gör en (1) enskild sak.

Artemis använder sig av komponenter och system, där komponenter håller data och systemen bearbetar dom. Ett system använder endast några få komponenter och arbetar endast på objekt som har dom komponenterna.

Planeterna ska ha följande komponenter:

- Position
- Hastighet
- Acceleration
- Massa
- Storlek
- Färg

Det kommer behövas system för bland annat:

Gravitation vilket behöver använda *position*, *massa* och *acceleration* för att räkna ut gravitations krafter mellan planeterna.

Acceleration vilket behöver *acceleration* och *hastighet* för att förändra hastigheten.

Hastighet vilket behöver *position* och *hastighet* för att uppdatera positionen.

Ritande vilket behöver *position* och *storlek* för att rita planeterna på skärmen.

5.2 B

Programmet använder de utomstående bibloteken libgdx och artemis.

Planeterna representeras av entities med följande komponenter som vardera håller ett object (t.ex. en vektor eller ett flytal).

- Position (vektor)
- Hastighet (vektor)
- Acceleration (vektor)
- Massa
- Radie

[1] http://libgdx.badlogicgames.com/features.html
[2] http://gamadu.com/artemis/index.html

- Färg (RGBA)

Fysiksystemen är:

Gravitationssystemet som beräknar gravitationsaccelerationen mellan planeter och summerar dessa för varje planet. Systemet uppdaterar alltså accelerationskomponenten för planterna.

Hastighetssystemet uppdaterar hastigheteskomponenten hos planeter med accelerationskomponenten.

Kollisionsystemet undersöker om och när planter kommer kollidera inom närmaste ticköch hanterar dessa som elastiskta stötar. Systemet är ansvarigt för att uppdatera positionskomponenten hos planeter.

Framtidssystemet beräknar hur en markerad planet kommer röra sig inom närmsta framtid och ritar detta på skärmen.

Positionssystemet används av framtidssystemet för att uppdatera planeters positioner med deras hastigheter. Kollisionsystemet är nämligen för långsamt för att framtidssystemet ska vara användbart.

Användargränssnitt-systemen är.

Inputsystemet hanterar markeringar, dragningar, zoomning och liknande. Systemet hanterar interaktion med simulator-världen.

Planet-ritar-systemet ritar planeter i simulatorvärlden.

Gränssnittssystemet hanterar vänstermenyn.

Template-gränsnittssystemet hanterar template-menyn.

Hjälp-gränsnittssystemet hanterar hjälp-menyn.

5.3 C

Det blev ganska många ändringar i hur programmet ser ut inuti.

Vissa system behövdes inte heller, som t.ex. accelerationsystemet vars uppgift utförs av gravitationsystemet. Kollisionsystemet ersatte på samma sätt Positionsystemet, vilket dock blev kvar för att användas av framtidssystemet.

Vidare delades ritningssystemet upp i flera system för att bl.a. enklare hantera interaktion från användaren (t.ex. mus- och tangenttryckningar).

6 Tekniska frågor

6.1 A

1. Vilken algoritm ska användas för att justera planeters position, hastighet och acceleration? Ska t.ex. alla planeter jämföras med alla andra planeter i universumet, eller ska algoritmen vara selektiv och inte räkna planeter för långt borta?

2. Hur hanteras kolissioner? Studsar, spricker eller smälts planeter samman? Hur kommer kollisionsystemet påverka resten av programmet? Behövs systemet snabbas upp?

3. Behöver programmet snabbas upp med parallellisering? I så fall, på vilket sätt?

4. Bör programmet kunna förutse hur planter kommer röra sig (för att t.ex. visa användaren omloppsbanor). Kan detta implementeras så att det inte kräver för mycket resurser?

6.2 B

Hur vi svarade på ovanstående frågor:

1. Förflyttningen av planeter har delats upp i system och komponenter som beskrivet i 5 Programdesign.

   Gravitationssystemet använder den s.k. Barnes-Hut-simulations-algoritmen [3] [4] [5] för att beräkna accelerationer. algoritmen bygger på ett s.k. Quad Tree, som är likt ett binärt sökträd, men med fyra undernoder. Varje nod motsvarar dessutom en kvadrat i spelvärlden och dess undernoder kvadranter av denna. Planeter spars endast i löven av trädet, och varje löv får endast innehålla en planet. Varje nod spar den totala massan av alla planeter den och alla dess undernoder håller och beräknar masscentrum.

   När trädet har byggts jämförs varje planet med trädet. Om en nods masscentrum är tillräckligt nära planeten dyker algoritmen ner i trädet, annars kan noden beräknas som en egen planet.

   Accelerationen från en planet till en annan beräknas med formeln $a = G * M/d^2$, där a är accelerationen, G gravitationskonstanten, M motstående planets massa (dvs. dit accelerationen är riktad) och d är avståndet mellan planeterna.

   Hastigheten uppdateras från massan likt `v += a*t` och positionen från hastigheten likt `p += v*t`. Där `t` är tid sedan förra uppdateringen.

   Kollisionsystemet använder en s.k. *Sweep and Prune*-algoritm (SAP) [6] [7].

   Till varje planet skapas en *hastighetscirkel* - den har samma position som planeten, men dess radie är planetes radie adderat med längden av planetens hastighetvektor, likt: $c.r = p.r + p.v * t$. Cirkeln motsvarar alltså det område planeten kan påverka inom närmaste uppdatering.

   HastighetsCirkeln passas sedan in i en fyrkant - dvs. max och min värden beräknas i x- och y-led. Hastighetscirklarna sorteras sedan efter deras x-min.

   Det är sedan lätt att kontrollera vilka cirklar som överlappar i x-led.

   Om två cirklar överlappar i x-led, kontrolleras de i y-led. Om de även där överlappar kontrolleras om och hur planeterna faktiskt kolliderar - genom beräkna tiderna de kolliderar.

   Alla kollisioner som sker inom t tid samlas och sorteras.

[3] http://en.wikipedia.org/wiki/Barnes-Hut_simulation
[4] http://arborjs.org/docs/barnes-hut
[5] http://www.cs.princeton.edu/courses/archive/fall03/cs126/assignments/barnes-hut.html
[6] http://jitter-physics.com/wordpress/?tag=sweep-and-prune
[7] http://www.codercorner.com/SAP.pdf

Den tidigaste kollisionen och de som inträffar *samtidigt* (enl. passande epsilon-skillnad) tas ut ur listan. Alla planeter uppdateras till tiden vid kollisionen likt `p += v * T`.

Kollisionerna hanteras och motsvarande planeters hastighetscirklar och dyl. uppdateras.

På detta sätt utförs alltså `p += v*t` stegvis och kollisionsystemet ersätter positionsystemet.

2. Kollisioner hanteras som elastiska stötar. Kollisionsystemet har snabbats upp med SAP som beskrivet ovan. Två kollisionshanteringsätt hade provats innan: alla planter jämförs med alla ($O(n^2)$), och en trädliknande struktur lik den i gravitationsystemet (troligtvis $\Omega(n \log n)$.

3. Parallellisering har endast används i framtidssystemet. Uppdatering av accelerationer, hastigheter och positioner körs i lagom takt genom att beräkna med tidsskillnader (t som nämnt ovan). Om motsvarande system skulle parallelliseras skulle dett troligtvis göras internt inom systemet (dvs. systemet skapar trådar och väntar på att dessa ska bli klara), vilket inte säkert hade gjort systemet snabbare eftersom dessa beror till stor del på sekventiella beräkningar. Parallelliseringen hade därför troligtvis bestått av få trådar som exekverar en stor del av systemet var, eller av många trådar som exekverar en mycket liten del av systemet.

   I det tidigare hade det troligtvis körts för få trådar för att det skulle gå snabbare och den senarer troligtvis för många trådar för att det skulle gå snabbare (och variabler hade behövt låsas för att användas av alla).

4. Framtidssystemet har implementerats, men utan kollisioner då dessa gjorde det för långsamt.

6.3 C

Det var inga större problem med de tekniska frågorna. Kollisionsystemet blev inte så snabbt som förhoppats, men fungerar ändå för ett ganska stort antal planeter.

7 Arbetsplan

7.1 A

Arbetet pågår under vecka 17 t.o.m. 20. Under denna tid är det 3 övningstillfällen: fre 26 april (v17), fre 3 maj (v18) och ons 15 maj (v20).

Projektplanen har deadline vid den första övningen. Slutrapporten och programmetkoden har deadline vid den sista övningen. Vid varje övningstillfälle ges en muntlig lägesrapport.

Vecka 17 spenderas på att skapa ett git-repository med ett kodskelett och att skriva projektplanen. Om det finns tid över läggs den på att till viss del att börja fylla kodskelettet.

Vecka 18 ska användas till att skriva så mycket av programmet som möjligt. Om det finns tid över börjar användartestningarna v18 och fullgörs annars i början av v19.

Vi räknar med ett fullt fungerande, debuggat och användartestat program i mitten av v19, ons 1 maj. Slutrapporten ska skrivas så mycket som möjligt parallellt med resten av arbetet, men framförallt slutet av vecka 19 används till rapporten. Om det finns tid över kan det användas till debugging och finslipning av programmet.

Vi räknar med att slutrapporten är klar i slutet av vecka 19, fre 11 maj.

Vecka 20 (totalt 2 dagar innan inlämning) används endast om något moment tog längre tid än förväntat.

Vi har ingen specifik arbetsfördelning utan skriver båda i koden och rapporten - däremot kanske inte i samma filer samtidigt. Vi håller kontakt via internet och kan snabbt komma överens om vad som behöver göras för tillfället och vem som gör vad.

7.2 B

Planen lämnades in i tid och programmet blev fungerande ganska snart, men kodandet fortsatte in i v20.

Första användartestet gjordes onsdag v19. Rapporten påbörjades först lördag v19.

Arbetsfördelning varierade men i sin helhet skrev Johannes större delen av fysiksystemen (förutom framtidssystemet) och Anton större delen av grafiken, programgrunden och framtidssystemet. Båda skrev dock lite på allt.

Vi skrev ungefär lika mycket på planen och rapporten.

7.3 C

Ett fungerande program var klart ganska tidigt, men att lägga till och förbättra tog tid.

Rapporten började därför på efter dess tänkte deadline och vi hann bara med ett användartest.

8 Sammanfattning

Av projektet har vi, förutom att ha fått en inblick i just de bibloteken vi använde, lärt oss om spel-fysik och -rendering och hur lite om hur dessa kan optimeras.

Vi har också fått en känsla för hur användainteraktion bör vara - t.ex. vilka knappar och musrörelser som är intuitivt.

Programmet kan byggas ut nästan hur mycket som helst, och vi skulle bl.a. gärna vilja lägga till lättare sätt för att skapa omloppsbanor, dvs. att användaren säger att en planet ska orbitera en annan och att start-parametrarna automatiskt justeras.

Mycket i spelet skulle också kunna förbättras och debuggas. Johannes vill t.ex. fortsätta optimera kollisions- och gravitationssystemen.

Tillsist har vi också lärt oss i hur mycket nytta man kan ha av användartestning.