TDDC74 - Projektspecifikation

Projektmedlemmar:
Gustav Danielsson gusda320@student.liu.se
Marcus Dahlqvist marda648@student.liu.se

Handledare:
Jonas Wallgren jonas.wallgren@liu.se

17 april 2016

Innehåll

1 Projektplanering

Projektet går ut på att skapa ett actionspel där två eller fler spelare tävlar mot varandra. Spelet bygger på att med hjälp av tangentbordet styra ett flygplan på vald tvådimensionell spelbana. För att vinna skall man skjuta ner motståndarens flygplan och all information uppdateras grafiskt i realtid till de båda spelarna.

1.1 Kort projektbeskrivning

Programmet bygger på ett tvådimensionellt koordinatsystem med x och y axel. Objektens rörelse beskrivs med vektorer enligt linjäralgebraiska ekvationer där deras hastighet och riktning ska kunna vara påverkbara. Kollisionerna bygger på att få objekten att detektera om de är nära nog ett annat objekt då de aldrig kommer dela samma flytvärde. Skulle ett objekt vara för nära aktiveras eventet för kollision som att planet störtar eller plockar upp en form av buff.

Alla objekt ska även vara påverkade av ett gravitationsfält som läggs över koordinatsystemet som får att alla objekt kommer att falla mot marken om denna kraft inte motverkas i form av lyftkraft i någon form. Detta leder till att om man åker rakt upp för länge kommer spelaren tappa kontrollen över flygplanet och störta för en mindre tid baserat på gravitationskraften, tills det att hastigheten har stigit igen.

Programmet ska kunna hantera projektiler som ska kunna skjuta ner motståndare men även andra objekt som hindrar spelarens framfart. Dessa projektiler ska inte vara påverkade av gravitation. Detta är för att underlätta siktandet men även har en projektil i ett realistiskt scenario inte är märkbart påverkad av gravitation på så korta distanser.

Spelet ska även innehålla olika former av bonusar som kan plockas upp. Detta ger allt från en fartökning till nytt vapen eller liknande. Dessa ska kunna påverka planets beteende och dess rörelse i koordinatsystemet. Även ska objekt kallat entities, som kan komma i form av till exempel fåglar eller luftballonger, kunna bli nedskjutna som kommer innehålla bonusar eller bara behöver bli förstörda så att man inte kolliderar med dem. Planet i sig ska kunna flyga i 16 olika riktningar för att ge en mjukare svängning samt ge fler riktningar att skjuta i så att spelaren har mer möjligheter att skjuta ner motståndaren. Vi svängning ska planet följa riktningen men även vridas runt

x axeln för att ge en visuellt mer realistisk bild av att plant svänger.

Slutligen ska spelets inställningar kunna påverkas via en huvudmeny. Inställningarna består av val av spelbana, flygplan, antal poäng till vinst mm. Samt så ska spelet kunna pausas när en knapp trycks ned, och en pausmeny öppnas. Från denna meny ska man kunna återgå till spelet, avsluta pågående omgång eller avsluta programmet helt.

1.2 Utvecklingsmetodik

Vi strävar efter att genomföra det huvudsakliga arbetet i skolan på de erhållna tiderna. Vid behov försöker vi främst utnyttja lediga tider under arbetsveckan med vid behov även göra eget arbete hemma. Då båda medlemmarna har tillgång till laptop kan arbetet ske på valfri plats. Utöver det så ses vi dagligen samt har god kontakt vilket gör att kommunikation mellan medlemmarna kan ske utan stora förhinder.

Koden kommer att versionshanteras med tjänsten gitlab som kommer att underlätta konflikter med koden samt så erbjuder gitlab möjligheten att gå tillbaka till äldre versioner av koden vid olösliga problem.

1.3 Grov tidplan

I början kommer vi huvudsakligen att arbeta tillsammans för att få en bra struktur och en gemensam grundidé att arbeta utifrån. Vi strävar att få det gemensamma arbetet över så fort som möjligt då det går effektivare att arbeta individuellt. Sen utifrån grunden kan vi börja jobba mer individuellt på kommandon och olika grafiska delar som planens utseende till menyer eller finjustera fysikmotorn. Vi kommer dock ses regelbundet för att få en bättre bild av vad den andra personen gjort. På grund av att vi ses dagligen blir det även lätt att diskutera eventuella konflikter som uppstår vid individuellt arbete.

Vi kommer behöva bekanta oss med att arbeta med rackets inbyggda grafikpaket för att skapa menyer och spelet i sig. Samt så behöver vi lära oss att arbeta med realtidsuppdateringar och inte skapa ett spel som upplevs allt för ryckigt i manövreringen.

Till halvtidsmötet ska fysikmotorn och rörelsen i koordinatsystemet vara klart så spelet är fungerande fast utan grafik. Efter mötet kommer vi fokusera på att implementera grafiken och sedan relevanta funktioner när spelet har

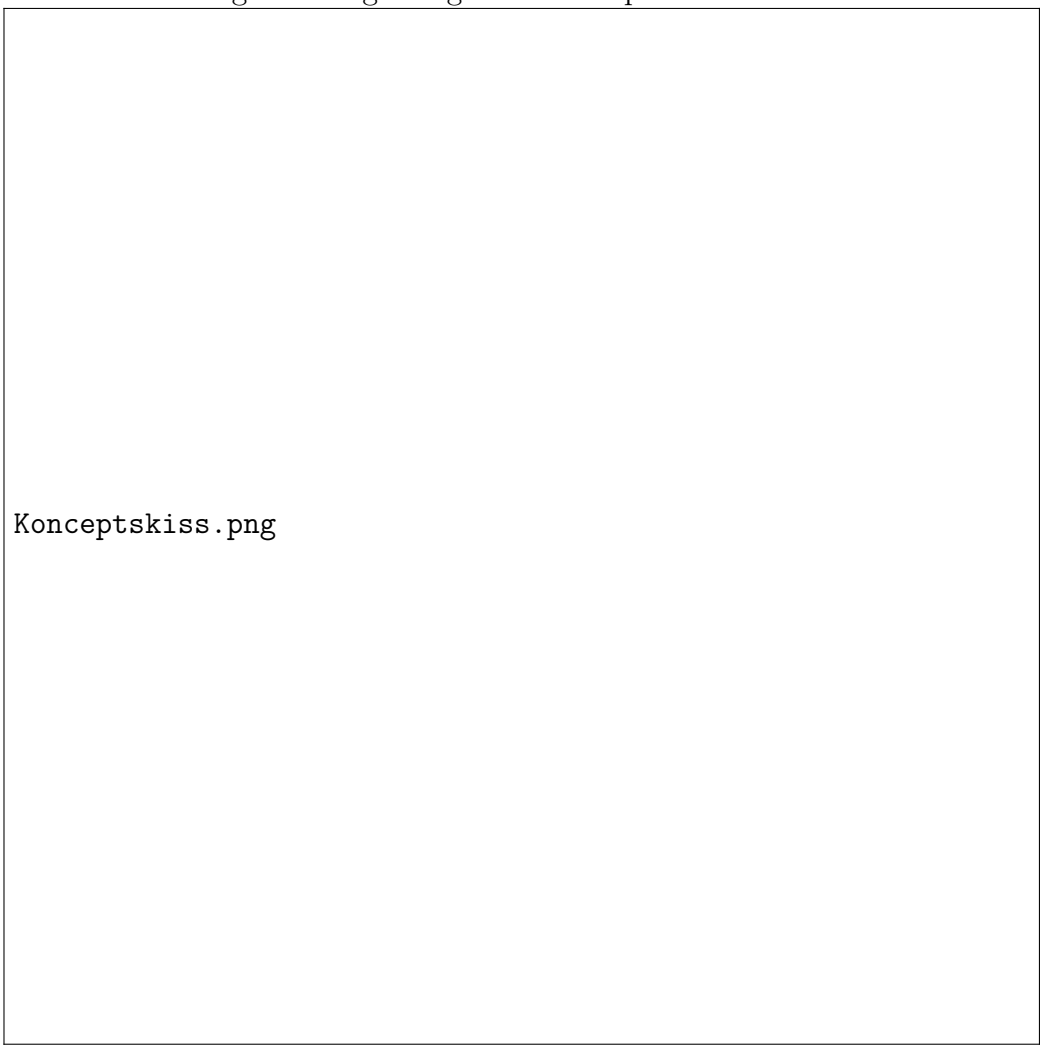
en grafik, såsom menyer, att kunna påverka vinstkriterier och avsluta spelet från en pausmeny.

1.4 Betygsambitioner

Då båda är intresserade av att göra ett så bra och givande projekt som möjligt siktar vi på att nå betyg 5. Vi båda arbetade på samma ingenjörsprojekt som byggde på att skriva ett reglertekniskt program och vi vill förbättra våra förmågor i utvecklandet av program ur ett projektperspektiv.

2 Konceptskiss

Figur 1: Ungefärlig skiss över spelets utseende.



Figur 1 ovan är en ungefärlig bild över det slutgiltiga spelets utdata. Spelarna(flygplanen) kommer att kunna styra sin riktning i kordinatsystemet genom att rotera medsols eller motsols. Utöver detta kommer man även kunna avfyra projektiler(gula prickar vid det vänstra planet), samt använda en temporär hastighetsökning(röd/gul mätare längt ned till vänster indikerar kvarvarande tid). Det kommer att vara möjligt skjuta till exempel en fågel eller ett flygplan, som i sin tur kommer att släppa någon form av bonus(lila objekt med blixt) som spelaren kan använda. Det finns även en indikator på

vilka buffs om är aktiverade(längst ned till vänster). Man kan även se hastighetsmätaren(längst ned till vänster), med ett rött område som indikerar när man riskerar att störta på grund av för låg hastighet. Under hastighetsmätaren finns en indikator på hur lång tid det är kvar tills det att nästa hastighetsbuff kan användas. På markytan finns 4 landningsbanor, och det är här spelarna kommer att återuppstå ifall de blivit nedskjutna. Höjden av flaggan ovanpå representerar antalet liv som kvarstår för spelaren. Man kan även se taket för spelplanen, och kollision med denna kommer att göra att planet börjar störta. Sist så kan man även se att det finns en menyknapp som pausar spelet med en meny.

Notera här att detta bara är ett exempel på hur det skulle kunna se ut, och att det inte nödvändigtvis kommer att vara på detta sätt vi representerar aktiverade bonusar, hastighet, antalet liv, placering av menyknapp etcetera. Till exempel så kan det även vara möjligt att spelet blir bättre om bonusar inte syns, att landningsbanorna är placerade annorlunda, att menyknappen är på något annat ställe eller liknande, vilket kommer ge sig alltefter funktioner blir implementerade.

3 Kravlista

Här specificeras kraven för projektet. De lägre kravprioriteringarna 4-5 är inte nödvändiga, om implementeras snarare efter tillgänglig tid efter 1-3 är färdiga.

Krav	Prioritet	Kategori	Kommentar
Världen ska byggas på ett tvådimensionellt koordinatsystem	1	Fysikmotor	
Det skall finnas objekt som kan ändra sin position i koordinatsystemet	1	Fysikmotor	
Positioner skall kunna uppdateras i realtid	1	Fysikmotor	

Objekt skall kunna förflyttas rakt i flera olika riktningar (16+)	1	Fysikmotor	Antal riktningar beror på vad som ger bäst spelupplevelse
Objekt skall kunna ha olika storlek	1	Fysikmotor	
Objekt skall kunna kollidera med varandra	1	Fysikmotor	Exempelvis aktivering av en buff eller krascha mot marken
Hastigheten för objekt skall vara varierbar	2	Fysikmotor	
Ett objekt som kolliderar med vänster/högerkant skall kunna komma fram på motsatt sida	3	Fysikmotor	
Enkel gravitiation skall kunna simuleras	3	Fysikmotor	Kommer ungefär fungera så att objekt som rör sig uppåt kommer att få en lägre hastighet än de som rör sig nedåt. Kommer ej vara en konstant acceleration nedåt.
Spelet skall kunna simulera vind	5	Fysikmotor	
Det skall vara möjligt att välja olika världar med annorlunda fysik	5	Fysikmotor	T.ex. välja att spela på månen med lägre gravitation eller liknande
Det skall finnas spelarobjekt som kan styras av spelaren	2	Spelelement	

Det skall finnas en möjlighet för två spelare	2	Spelelement	
Det skall finnas icke-spelarobjekt som spelaren kan interagera med	2	Spelelement	
Det skall finnas ett markplan längst ned i världen	2	Spelelement	Vissa element kommer att elimineras vid kollision med marken
Det skall finnas en himmel som gör att spelaren faller	2	Spelelement	Fungerar som ett tak fär spelplanen
Spelaren skall kunna avfyra projektilobjekt	2	Spelelement	Fungerar som vapen och kommer bland annat att kunna förstöra motspelare
Det ska finnas objekt som spelaren kan kollidera med som gör att spelaren kraschar	2	Spelelement	
Det skall finnas ett vinst/förlustkriterie	2	Spelelement	I sin grund att skjuta ned motspelaren
Vid för låg hastighet riktad uppåt skall spelarobjektet kunna förlora kontrollen	2	Spelelement	Planet faller rakt ned utan kontroll till en viss hastighet uppnås, varvid spelaren kan börja styra igen
Om två spelare kolliderar med varandra skall båda krascha	3	Spelelement	

Det skall finnas en plats där spelaren kan återuppstå	3	Spelelement	Kommer att vara i form av en landningsbana vid marken
Vinst/förlustfunktionen ska utökas så att varje spelare har flera liv	3	Spelelement	
Spelaren skall vara immun mot skada en kort tid efter återuppståndelse	3	Spelelement	För att undvika att motspelaren missbrukar den överlägsna positionen
Spelaren skall ha tillgång till en aktiverbar hastighetsboost	3	Spelelement	Kommer att ha en tidsbegränsad aktiveringsfrekvens, som eventuellt kan förändras av bonusar
Det skall finnas bonusar som spelaren kan aktivera	3	Spelelement	Kommer att göra saker som att öka maxhastighet, skjuthastighet m.m.
Vid krasch ska spelaren störta mot marken, fortfarande med kollision	4	Spelelement	Det vill säga att andra objekt fortfarande kan kollidera med den till den når marken
Det skall finnas stöd för upp till 4 spelare	4	Spelelement	
Det skall vara möjligt att välja olika vinstkriterier	5	Spelelement	Detta skulle kunna vara: Tidsbegränsningar, fler liv, lagspel för 4 spelare.
Man skall kunna starta spelet med 1 kommando	1	Användargränssnitt	Kommandot skall öppna en startmeny

Det skall finnas en startmeny	2	Användargränssnitt	Skall innehålla instruktioner, start av spel och senare möjlighet att justera spelinställningar
Det skall finnas en undermeny för instruktioner	2	Användargränssnitt	Här visas kontroller, vilka buffs som finns, och hur spelet går till
Det skall finnas en knapp som startar huvudspelet i menyn	2	Användargränssnitt	
Det skall finnas en knapp som avslutar spelet i huvudmenyn	2	Användargränssnitt	
Huvudspelet skall kunna styras endast med ett tangentbord	2	Användargränssnitt	Kontrollerna som finns kommer att vara: Ändring av färdelseriktning medsols och motsols, avfyrning av vapen och aktivering av hastighetsboost
Det skall finnas en knapp som avslutar spelet i huvudspelet	2	Användargränssnitt	
Det skall finnas en knapp som pausar spelet	3	Användargränssnitt	i denna meny skall man kunna avsluta spelet eller återvända till huvudmenyn
Menyn skall kunna navigeras endast med mus	3	Användargränssnitt	

Det ska vara möjligt att själv ändra kontrollbindningarna i menyn	5	Användargränssnitt	
Huvudspelet skall vara grafiskt implementerat	2	Grafik	
Alla menyer skall vara grafiskt implemenerade	3	Grafik	
Det skall finnas en indikator för antalet kvarstående liv	3	Grafik	
Det skall finnas en indikator för hastighet	4	Grafik	
Det skall finnas en indikator för kvarvarande tid till hastighetsbonus	4	Grafik	
Det skall finnas en indikator för aktiverade bonusar	5	Grafik	
Det skall finnas stöd för ljud i spelet	4	Övrigt	
Ljud skall implementeras i huvudspelet	4	Övrigt	
Det skall finnas musik i menyerna	5	Övrigt	
Vid spelets slut skall statistik för matchen visas	5	Övrigt	Antal avfyrade skott, antal liv vid spelets slut, tidsåtgång, aktiverade bonusar, m.m.

4 Implementation

Här beskrivs ungefärligt processen under ett programvarv, samt de datatyper som kommer att användas av programmet.

4.1 Processbeskrivning

Spelet kommer att uppdateras i realtid med någon lämplig uppdateringsfrekvens, där vilken framkommer med framtida tester. Vid en uppdatering kommer först alla objekt i rörelse att förflyttas beroende på deras riktning och hastighet. Steget efter detta är att detektera om några kollisioner har skett. Eftersom koordinatsystemet inte är i form av ett rutnät, så kommer positionerna istället komma att beskrivas av flyttal. Då sannolikheten att två objekt befinner sig på exakt samma flyttal vid samma tidpunkt är väldigt liten, så kommer objekten istället att ha en storlek, vilket innebär att kravet för kollision snarare blir att man jämför avståndet mellan två objekt och om den sammanlagda storleken är större än detta avstånd räknas det som en kollision.Om en kollision väl detekteras, kommer dess effekt att igångsättas efteråt, vilket till exempel skulle kunna vara aktivering av en buff eller en krasch.

Efter det att kollisioner har blivit behandlade, är nästa steg vara att avläsa spelarnas kommandon. Efter avläsning kommer sedan dessa värden sparas. De kommandon som finns är rotation medsols/motsols, avfyrning av projektil samt aktivering av hastighetsbuff. Värdena kontrolleras sedan för att se om de är tillåtna, vilket till exempel skulle kunna vara att spelaren nyss avfyrat en projekttil eller att spelaren störtar, och beroende på detta så kommer en åtgärd att utföras eller inte.

Nästa steg är att se om några nya objekt ska skapas, vilket antingen är en spelare som återuppstår eller ett icke-spelarobjekt som till exempel en fågel eller en buff. Objekt som spelare och bonusar kommer att skapas vid vissa händelser, medan fåglar och liknande kommer att skapas med någon form av slumpalgoritm.

Efter detta är ett programvarv färdigt för fysikmotorn, och det som är kvar är att uppdatera grafiken. Detta görs dock inte varje programvarv, utan kommer att göras med en frekvens på 60 Hz, och om tiden för detta inte är uppfylld så väntar programmet på att fysikmotorn ska uppdateras igen.

4.2 Abstrakta datatyper

I följande underrubrik beskrivs de olika datatyper och klasser som används i programmet.

4.2.1 Flying-unit

Datatypen **flying-unit** kommer att vara en överklass till alla flygande objekt, det vill säga spelare, hinder(exempelvis fåglar) och bonusar. Denna detatyp kommer även att representera projektiler som spelaren avfyrar, men då dessa projektiler inte har några extra egenskaper utöver de som beskrivs av **flying-unit**, så kommer denna klass att användas för att beskriva dem direkt.

Metod	Variabel	Beskrivning
get-size	-	Returnerar bonusens storlek.
get-coordinates	-	Returnerar var spelaren befinner sig just nu.
get-direction	-	Returnerar åt vilken riktning spelaren är riktad åt i radianer.
get-speed	-	Returnerar vilken hastighet spelaren rör sig i.
set-coordinates!	**X Y**	Ändrar spelarens koordinater.
set-direction!	**angle**	Ändrar spelarens riktning.
set-speed!	**speed**	Ändrar spelarens hastighet.

4.2.1.1 Player

Datatypen **player** kommer att vara en klass som beskriver spelarobjekt, och kommer utöver de allmänna variablerna hos flygande objekt även att innehålla spelarobjektets unika egenskaper.

Metod	Variabel	Beskrivning
get-size	-	Returnerar spelarens storlek
get-max-speed	-	Returnerar spelarens tillåtna maxhastighet.
get-lives	-	Returnerar spelarens antal kvarstående liv
is-defeated?	-	Returnerar #t om spelaren blivit besegrad, det vill säga inga liv kvar, och annars #f.
can-fire?	-	Returnerar #t om spelaren är tillåten att skjuta, annars #f.
can-speedboost?	-	Returnerar #t om spelaren är tillåten att använda speedboost, annars #f.
is-plunging?	-	Returnerar #t om spelaren har förlorat kontrollen på grund av för låg fart, eller kolliderat med toppen av spelplanen, annars #f.
is-dead?	-	Returnerar #t om spelaren är ur spel på grund av krasch, annars #f.
set-max-speed!	**speed**	Ändras spelarens tillåtna maxhastighet.
set-lives!	**lives**	Sätter spelarens antal liv.
toggle-defeated!	-	Växlar om spelaren har blivit besegrad.
toggle-fire!	-	Växlar om planet är tillåtet att skjuta.
toggle-speedboost!	-	Växlar om planet är tillåtet att använda speedboost.
toggle-plunging!	-	Växlar om planet har förlorat kontrollen.
toggle-dead!	-	Växlar om spelaren är ur spel.

4.2.1.2 Buff

Datatypen **buff** kommer att ara en klass som beskriver bonusobjekt, och kommer utöver de allmänna variablerna för flygande objekt även beskriva vilken typ av bonus det är.

Metod	Variabel	Beskrivning
get-bonus-type	-	Returnerar vilken typ av bonus objektet är.

4.2.1.3 Entities

datatypen **entities** kommer att vara en klass som beskriver flygande objekt som inte styrs av spelaren(exempelvis fåglar), och kommer förutom de allmäna egenskaperna för flygande objekt att beskriva de unika egenskaperna för dessa objekt.

Metod	Variabel	Beskrivning
get-buff	-	Returnerar vilken typ av bonusobjekt som detta objekt lämnar efter sig vid kollision med en projektil, eller #f om ingen lämnas.

4.2.2 World

Datatypen **world** kommer att vara en klass som beskriver alla icke-rörliga kollisionsobjekt på spelplanen, det vill säga marken, himlen och landningsbanorna.

Metod	Variabel	Beskrivning
get-coordinates	-	Returnerar vilka koordinater objektet har.
get-size	-	Returnerar objektets storlek.
get-type	-	Returnerar typen av kollisionsobjekt. Detta för att avgöra vilken åtgärd som ska utföras vid kollison, det vill säga förlust av kontroll för himmlen och krasch för mark och landningsbana.

4.2.3 HUD

Datatypen **HUD** kommer att beskriva de objekt som ger information till spelaren, det vill säga hastighetsmätare, buffräknare, flaggor och hastighetsbuffindikator. Dessa skall kunna uppdateras i realtid, och beroende på vad som händer så skall bilderna både kunna förflyttas(visaren på hastighetsmätaren) eller ändras helt(antalet bonusar),

4.2.4 Background

Datatypen **background** kommer att beskriva den bakgrund vilket spelet spelas över, och även bakgrunden i menyerna. Denna kommer att vara statisk under spelets gång, och kommer därför bara att behöva uppdateras vid uppstart och när man går till menyn.

4.2.5 Button

Datatypen **button** kommer att representera alla tryckbara knappar i menyerna och eventuellt i huvudspelet. Dessa ska utföra kommandon om spelaren trycker på detta område av skärmen med musen.

4.2.6 Vector

Datatypen **vector** kommer att beskriva vektorer och punkter i koordinatsystemet, om kommer bestå av ett par där car-delen är x-värdet och cdr-delen y-värdet.

4.2.7 Matrix

Datatypen **matrix** kommer att beskriva matriser som används för beräkningar, och kommer att bestå av en lista, där första elementet är en lista av första raden i matrisen, andra elementet är en lista av andra raden, etcetera.

4.3 Testning

När det gäller tester så är det främst viktigt att rörelserna i koordinatsystemet och kollisiondetekteringen fungerar på rätt sätt. Vi kommer därför att behöva testfunktioner som avläser och skriver ut de variabler som framkommer vid ett programvarv, för att se att saker får rätt position, att kollision detekteras korrekt samt att händelser kontrolleras och utförs korrekt. Lämpligt här är att skapa en testfunktion där programmet kan uppdateras med hjälp av ett knapptryck eller kommando istället för en klocka, för att på så sätt kunna analysera varje uppdatering på detaljnivå. Man bör även i kunna sätta alla variabler manuellt i testfunktionen, för att kunna testa specifika fall enklare.

Tidigt i implementeringen kommer vi främst att testa så att rörelser fungerar korrekt, så att föremål får rätt position i koordinatsystemet givet en riktning och en hastighet. När detta fungerar är huvudfokus att testa kollision av objekt och i sin tur att storlekar av objekt fungerar på rätt sätt. Här finns det även viktiga speciallfall att testa, till exempel om 3 objekt kolliderar med varandra under samma programvarv, då måste någon form av händelseprioritering göras.

Senare i projektet kommer även grafiken att behöva testas, men då fysikmotorn då redan ska vara implementerad handlar detta främst om att se så att kollisioner och liknande inträffar synkroniserat med att de grafiska objekten kolliderar.

5 Tidsrapportering

5.1 Gustav Danielsson

Vecka	**Arbetsuppgift**	**Tid(h)**
14	Skrivit kravspec	4
Tot:	Summerad tid	4

5.2 Marcus Dahlqvist

Vecka	**Arbetsuppgift**	**Tid(h)**
14	Skrivit kravspec	8
Tot:	Summerad tid	8