NTNU
Det skapende universitet

Bruk av spillmotor for utvikling av undervisningsspill

Lars Vråle

Master i teknisk kybernetikk
Oppgaven levert: August 2009
Hovedveileder: Geir Mathisen, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

1. Sette seg inn i teorien rundt spillmotorer.
2. Velge en spillmotor som kan benyttes sammen med java applets.
3. Designe et testsystem for spillmotoren
4. Så langt det rekker, implementere designet.

Oppgaven gitt: 12. januar 2009
Hovedveileder: Geir Mathisen, ITK

Forord

Denne oppgaven er skrevet som en avsluttende oppgave på masterstudiet ved Norges teknisk-naturvitenskapelige universitet (NTNU), på institutt for teknisk kybernetikk. Oppgaven er igangsatt av Cyberlabs, etter et ønske om å kunne utvikle 3-dimensjonale spill som illustrerer fysiske prinsipper. Jeg har valgt denne oppgaven på grunn av min interesse for datamaskingenerert grafikk. I tillegg synes jeg det er interessant å kunne illustrerere de fysiske prinsippene som man lærer om på skolen. Gjennom oppgaven har jeg fått kjennskap til hvordan spillmotorer fungerer, fått erfaring med å benytte 3D-modelleringsverktøy og ikke minst fått mer erfaring med å programmere Java. Jeg vil gjerne takke veilederen min Geir Mathisen for hans tålmodighet med meg gjennom semesteret. Jeg vil også takke Tor Ivar Eikaas i Cyberlabs for en spennende oppgave, og Jørn Bersvendsen i firmaet for oppfølging i oppgaven Videre vil jeg takke Stein Løtveit og Thomas Solmund Næss Wollbekk for hjelp og forklaringer om hva en spillmotor er. Jeg vil også takke moren og faren min for moralsk støtte gjennom hele studieperioden.

Trondheim, 3. August 2009

Lars Vråle

Sammendrag

Denne oppgaven er gitt av Cyberlabs som er et firma som arbeider med å utvikle spill og programmer for å illustrere matematiske modeller av fysiske prinsipper. Når elever skal lære seg matematikk og fysikk kan de ofte ha vanskeligheter med å knytte det de lærer til virkeligheten. Ved å gi en grafisk fremstilling av hva ligningene de lærer om betyr i praksis, kan de lettere forstå sammenhengen mellom matematikken og virkeligheten. Cyberlabs har tidligere i hovedsak benyttet seg av 2-dimensjonale spill, men ønsker gjennom denne oppgaven å se på mulighetene for å utvikle 3-dimensjonale undervisningsspill som kan kjøres som en Java applet i en nettleser.Dataspill er i dag komplekse dataprogrammer som krever mye behandling av blant annet grafikk og nettrafikk. I og med at disse egenskapene er felles for dataspillene, kan denne funksjonaliteten samles i egne moduler. Disse modulene kalles spillmotorer. I denne oppgaven skal det velges ut en spillmotor som er tilgjengelig som åpen kildekode. Motoren skal kunne kjøre som en Java applet, og det velges derfor en Java-basert spillmotor. Forskjellige spillmotorer blir vurdert, og gjennom å sammenligne de mest aktuelle motorene har jeg kommet fram til at jMonkey Engine er best egnet. Dette er sammenfallende med hva Cyberlabs har antydet, men de ønsket en grundigere vurdering slik som er foretatt i denne oppgaven. For å vurdere spillmotoren videre blir det i denne oppgaven også utviklet et testspill. Cyberlabs har et ferdig modellsystem for en tank med innløp og utløp, der nivået i tanken kan styres. jMonkey Engine benyttes for å lage et spill rundt dette modellsystemet. Det ferdige spillet finnes på en cd som er vedlagt denne oppgaven. Spillet brukes også til å evaluere jMonkeys egnethet, og det konkluderes med at motoren kan benyttes, men at det fremdeles må jobbes med blant annet bedre integrasjon av fysikkmotoren.

Innhold

Figurer

Tabeller

Kapittel 1

Innledning

Når elever og studenter i dag lærer seg matematikk, er det mange som ikke ser bruksområdene for det de lærer. Selv i fysikken blir ligningene ofte vanskelig å se i sammenheng med virkeligheten. En mulighet for å gi større forståelse for betydningen av matematikken, er å gi en grafisk fremstilling av hva ligningene faktisk betyr. Dette gir muligheten til å lage spill der oppgaven er å styre en fysisk prosess ved å endre parametre i ligningene som beskriver prosessen. Cyberlabs er en bedrift som har utviklet slike spill i et system de kaller PIDstop. PIDstop er en rekke produkter som tilbyr forskjellige lære-systemer. Disse systemene består blant annet av spill som presenterer simuleringer av forskjellige fysiske prinsipper. Ved å gi en visuell presentasjon av konsepter som studenter og elever kun kjenner som matematiske formler, kan man oppnå en bedre forståelse av fysikken bak. Pidstop systemet er utviklet i samarbeid med Norges tekniske-naturvitenskaplige universitet (NTNU). Disse spillene benyttet seg av 2-dimensjonell grafikk I dag er spill svært ofte 3-dimensjonale. Cyberlabs ønsker å ha mulighet til å utvikle spill i 3D. Dette kan gi større muligheter for hva som kan simuleres. I tillegg kan mer avansert grafikk gjøre spillene mer interessante for ungdom. 3D-dimensjonale spill er kompliserte å lage, og det benyttes derfor ofte systemer, som tilbyr enkle løsninger for å tegne denne grafikken, kalt spillmotorer. En rekke slike motorer er også lansert som åpen kildekode.
Denne oppgaven skal gi en forståelse av hva slike spillmotorer tilbyr. Den skal også utvikle et testspill som benytter seg av Cyberlabs sine simuleringssystemer, for å se om og hvordan en slik system kan fungere.

1.1 Definering av oppgaven

1. Sette seg inn i teorien rundt spillmotorer.
2. Velge en spillmotor som kan benyttes sammen med java applets.
3. Designe et testsystem for spillmotoren
4. Så langt det rekker, implementere designet.

1.2 Definisjoner

Denne oppgaven er skrevet på norsk, men omhandler et tema der mye av terminologien er på engelsk. Det er gjort en innsats i å oversette flest mulig av disse til norske ord. For å gi en en forståelse for disse ordene, og for å forklare en del av de engelske begrepene som benyttes i teksten har det i løpet av denne oppgaven blitt laget en liten oordliste. Tabell 1.1 inneholder forklaringer på ord og utrykk brukt i denne oppgaven og hva disse ordene er på engelsk.

Tabell 1.1: Definisjoner og forklaringer av ord og uttrykk

Ord i oppgaven	**Forklaring**	**Engelsk**
[X,Y,Z]	Et 3D-dimensjonalt rom	[X,Y,Z]
API	Application Programming Interface -et grensesnitt mot en programmodul	API
Bevegelsessløring	Utydeliggjøring av et objekt/-bilde på grunn av raske bevegelser	Motion blur
Bildefrekvens	Antall bilder som vises per sekund	Framerate
Biblioteksfil	En fil som inneholder funksjoner og metoder som kan brukes av andre programmer	Library
GPU	Prosessor for grafikk	Graphical Processing Unit
Himmelboks	Metode for å skape inntrykk av at rollefiguren har en større verden rundt seg	Skybox
Kartlegging	Lage en tabell over hvordan teksturer og annet skal legges over en 3D-dimensjonal figur.	Mapping
Mellomlager	Data som allerede er hentet kan lagres i et mellomlager så det kan hentes herfra neste gang det skal brukes	Cache
Myktlegeme	Gjenstand som kan deformeres	Soft Body
Node	Knutepunkt	Node
Objekt-transformasjon	Flytte, rotere eller skalere et objekt	Transformation

Fortsatt på neste side...

Tabell 1.1 – Fortsatt

Ord i oppgaven	Forklaring	Engelsk
Omgivelseslys	lys som treffer alle deler av en scene likt	ambient light
Pixel	Punkt på skjermen. Kommer av Picture Element"	Pixel
Primitiver	Enkle objekter somm kuber, kuler, kjegler osv.	Primitives
Render	Program som tegner opp grafikk, se avsnitt 2.1.3.2	Render
Rollefigur	Figur som en som spiller et spill kan styre i et spill	Character
Stabel	Datastruktur for midlertidig lagring av data	Stack
Stivt legeme	Gjenstand som ikke kan deformeres	Rigid Body
Subtre	En del av at tre	Subtree
Tekstur	Overflatestruktur. Et bilde av en overflate som kan legges på en flate.	Texture
Transformasjon	Flytte et objekt.	Transformation
Tre	Hierarkisk graf, der hver node kan ha en forelder, men mange barn	Tree
Wiki	Nettside der alle eller enkelte brukere kan endre innholdet. Brukt til oppslagsverk, bruksanvisninger osv.	Wiki

1.3 Oppgavens utforming og gjennomføring

Kapittel 2 av oppgaven tar for seg en rekke egenskaper som er viktige for spillmotorer, og 3D-spillmotorer spesiellt. Dette skal gi en innsikt i hva som bør kreves av den valgte spillmotoren. Det er også innhentet informasjon om en rekke åpen-kildekode spillmotorer. Kapittelet vil også gi en kort innføring i Java Applets og dets muligheter og begrensninger. I kapittel 3 gjøres en sammenligning av en rekke aktuelle spillmotorer, og en motor velges ut. Et testsystem som kombinerer spillmotoren og Cyberlabs simulator skal utvikles. Kravspesifikasjonene fra Cyberlabs for dette systemet presenteres. Det blir så utviklet tester for å kunne undersøke testsystemet. Deretter arbeides det frem et design for testsystemet. Dette testsystemet og hvordan prosessen frem mot dette designet har foregått blir presentert.

I kapittel 4 presenteres implementasjonen av designen fra kapittel 3. I kapittel 5 diskuteres systemet på bakgrunn av testene som ble satt opp i kapittel 3. Forslag til hvordan det utviklede programmet kan forbedres presenteres i kapittel 6.

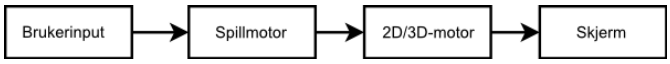
Kapittel 2

Spillmotorer og måter å bruke dem på nettsider

2.1 Spillmotorer

2.1.1 Hva er en spillmotor

Når man skal lage et spill, er det mange deler som må på plass. Før spillutviklerne kan begynne å bygge ut ideen bak spillet, må man ha funksjoner for å presentere grafikk på skjermen, ta imot spillerens tastetrykk, koble seg mot nettverk osv. En spillmotor er er en en programpakke som gir programmereren tilgang til de funksjoner som er felles for de fleste spill. På denne måten kan det fokuseres mer på det faktiske innholdet i spillet, i stedet for å bruke budsjett på å skrive alle program-rutinene selv for hver produksjon.[22] Spillmotorer ble tatt i bruk mot slutten av 1980-tallet. Spillene begynte på dette tidspunktet bli så kompliserte at spillfirmaene begynte å lage script-systemer som skulle forenkle programmeringen. For eksempel utviklet spillfirmaet LucasArts scriptspråket SCUMM (Script Creation Utility for Maniac Mansion) for spillet Manic Mansion. Denne gjorde det enklere å legge inn grafikk, spille av musikk, registrer hva brukeren gjør osv. Utover 90-tallet ble spillene stadig mer avansert. Et av de store skrittene var det å gå fra 2D-grafikk til 3D-grafikk. 3D-grafikk er kompleks matematikk. Det var derfor naturlig å samle programrutinene for tegning av 3d-grafikk i en såkalt 3d-motor.[27]
En 3d-motor er en samling biblioteker med et API som gir utvikleren tilgang til funksjoner for å tegne 3d-grafikk på skjermen. 3D-motorens oppgave er å foreta alle lav-nivå oppgaver slik som rotasjoner,transformasjoner og å tegne bildet på skjermen. Spill-motoren sender signal til 3d-motoren om hva som skal tegnes.
Kort sagt virker et spill slik:



Figur 2.1: Enkel oversikt over en spillmotor

Ordet motor brukes her om et system som kan sees på som en "black-box". Utvikleren

sender inn inngangsverdier og mottar utgangsverdier, men er ikke interessert i hva som skjer inne i motoren. [16]
Spillmotorer er altså laget for å forenkle prosessen med å lage spill. Enkelte motorer går så langt at de blir kalt for "pek-og-klikk"-motorer. Med dette menes at spill kan lages i et GUI (grafisk brukergrensesnitt) der man kan velge inn moduler og laste inn modeller på en enkel måte. Dette gir en brukervennelig og enkel måte å lage spill på, der mengen programmering reduseres kraftig. Men denne enkelheten går også på bekostning av friheten. Spesielle krav til f.eks. grafikk og fysikk kan ikke alltid tilfredsstilles i slike motorer. Et eksempel på slike motorer er 2D-spillmotoren Adventure Game Studio.[1]
Mange spillmotorer gir istedenfor brukeren direkte tilgang til bibliotekene, og gir på denne måten stor frihet til hvordan spillet skal se ut og oppføre seg. For å øke brukervenneligheten kan det følge med forskjellige verktøy til spillmotorene, se avsnitt 2.1.3.16.

2.1.2 Åpen kildekode kontra kommersiell

Når man skal velge en spillmotor må man gjøre et valg på om man ønsker å kjøpe et kommersielt produkt, eller om man vil gå for en løsning med åpen kildekode. Kommersielle motorer har den fordelen at man kan forvente at motoren gjør det dokumentasjonen sier den skal gjøre, i og med at det er et produkt man har betalt for. Slike motorer har også gjerne god dokumentasjon og bør helst være brukervennelige, da produsentene skal selge produktet. Spillmotorer kan imidlertid være svært dyre.
Spillmotorer som er lagt ut som åpen kildekode har den fordelen at de er gratis. En annen fordel er at de ofte kan ha mange brukere, ettersom hvem som helst kan ta dem i bruk. Dette kan føre til at det er lett å få hjelp til å løse problemer som dukker opp. Siden kildekoden er tilgjengelig, er det også mulig å endre den selv, dersom man skulle ha behov for dette. Den største ulempen med disse motorene er at det ikke nødvendigvis følger med noen god bruksanvisning. Disse er ofte laget som wikier som blir vedlikeholdt av brukerne, og er da nødvendigvis ikke helt oppdaterte. Hvis motoren er oppdatert uten at wikien har blitt endret, kan denne til og med gi gale opplysninger. Disse spillmotorene er vanligvis underlagt visse lisenser som begrenser bruken av dem. Dette kan være lisenser som tillater privat bruk, men ikke kommersiell bruk, eller lisenser som gir full frihet til bruk. De følgende lisensene er mye i bruk:

2.1.2.1 GNU General Public License (GPL)

Dette er den mest brukte lisensen for åpen kildekode. Programvare som er utgitt under denne lisensen krever at kildekoden av programvaren også er tilgjengelig for alle brukere. I tillegg kreves det at dersom programvaren brukes som en del av av et annet program, må dette nye programmet også utgis under GPL-lisensen.[28]

2.1.2.2 GNU Lesser General Public License (LGPL)

Denne lisenstypen er en mindre streng utgave av GPL. Den viktigste forskjellen er at man kan koble programvare lisensiert under LGPL til et nytt program uten å lansere det

nye programmet under LGPL. Forutsetningen er at man lager programmet slik at det vil fungere med fremtidige oppdateringer av den LGPL-lisensierte programvaren. [29]

2.1.2.3 zLib

Denne lisensen er laget for å beskytte den orginale programvaren. Den krever at man ikke påstår å ha laget den opprinnelige programvaren. Man må ikke la det se ut som den modifiserte programvaren er den orginale programvaren, og lisensen må ikke fjernes fra kildekodedistrubisjoner. Lisensen krever ikke at kildekoden legges ved det distrubierte programmet.[35]

2.1.2.4 BSD

BSD lisensen krever at distrubiert kildekode inneholder den samme copyright-notisen, betingelsene osv. som den orginale prograavaren har. Det samme gjelder det kompilerte programmet. Dersom man benytter programmet må man dessuten ha fått tillatelse fra de som har laget det opprinnelige programmet.[26]

2.1.2.5 MIT

MIT (Massachusetts Institute of Technology) har laget en lisens som skal gjøre programvaren som ligger under den tilgjengelig til bruk for proprietær programvare. MIT lisensen tillater bruk av programmet, så lenge MIT-lisensen også blir lagt med i det ferdige programmet [33]

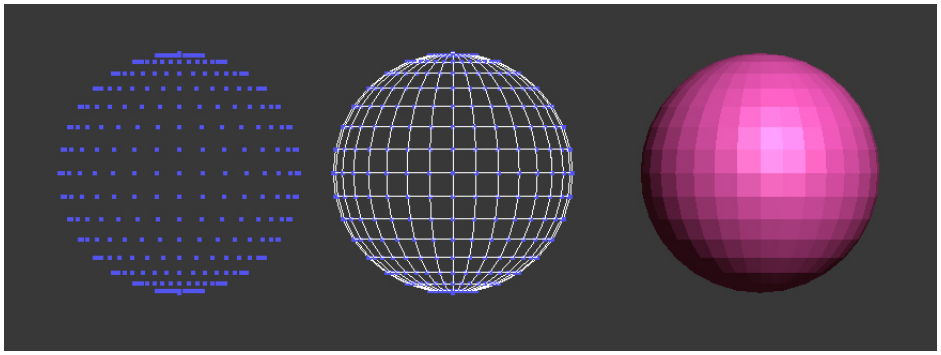
2.1.3 Generelle egenskaper hos spillmotorer

2.1.3.1 3D motor

Den åpenbare forskjellen på 2D og 3D er den ene dimensjonen. Siden grafikken skal prensenteres på en 2-dimensjonell skjerm er det intuitivt lettere med et 2D-spill. Her kan bakgrunner og bevegelser være tegnet på forhånd. Bakgrunnene kan her være enten et statisk bilde, eller en video. Enkeltobjekter og karakterer som skal røre på seg kan tegnes for seg selv, og plasseres/beveges på ønsket sted i bildet. Alle disse bildene blir lagret som pixel-tabeller. I steden for å forhåndstegne grafikken kan man benytte seg av vektorgrafikk. I vektorgrafikk beskrives grafikken ut i fra punkter, linjer og polygoner. Disse kan beskrives matematisk, og det er slik denne grafikken lagres. Her regner datamaskinen seg frem til hvordan linjene skal se ut før de presenteres på skjermen. Filstørrelsen på et vektorgrafikkbilde kan derfor bli vesentlig mindre enn et forhåndstegnet bilde. I tillegg kan alle deler innefor bildet endre størrelse, farge, rotasjon osv. uten tap av bildekvalitet. Når tegningene er beskrevet som matematiske formler, er ikke overgangen til 3D stor.

3D-Grafikk er i all hovedsak vektor-grafikk i et 3-dimensjonalt koordinat-system [X,Y,Z] 3D-Grafikk må lagres med to grunnverdier. Punkt-plassering i [X,Y,Z] og mellom

hvilke punkter man trekker en forhåndsdefinert/brukerdefinert vektor. Denne vektoren kaller vi en linje. En linje brukes til å definere rammene til et polygon. Et polygon håndteres som en flate som kan motta lys, farge og ferdigtegnede pixler(teksturer). Det er polygonene brukeren vil se gjengitt på skjermen. Objekter i [X,Y,Z] er samlinger av polygoner.

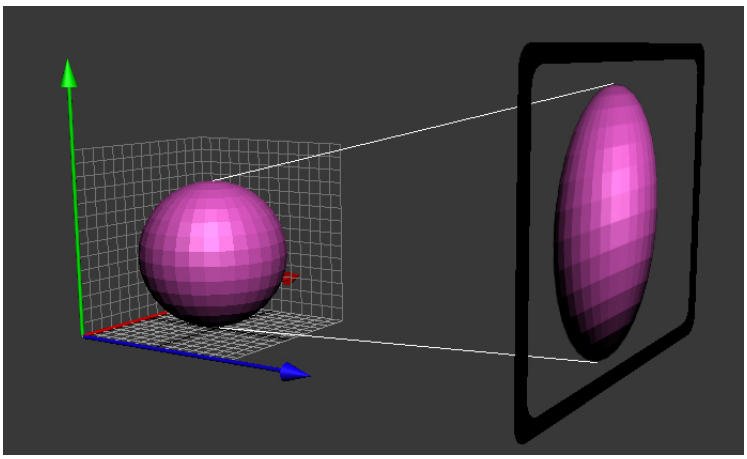


Figur 2.2: punkter, linjer og polygoner

3D-motoren håndterer all informasjon, beregning og tegning av 3D-grafikk. Oppgaven med å vise bildet på dataskjermen gjøres av en del av 3D-motoren som kalles renderen.

2.1.3.2 Renderer

En viktig del av en 3D-motor er renderen. Renderens oppgave er å presentere all informasjonen om scenen, alle objektene, lys og teksturer som et 2-dimensjonalt bilde, slik at det kan vises på datamaskinens skjerm. Når dataene er kalkulert må grafikken tegnes på skjermen for å være synlige. Siden skjermen er flat må grafikken regnes om til det skjermens 2-dimensjonale koordinatsystem. Denne prosessen kalles for transformasjon.



Figur 2.3: Transformasjon til skjerm-koordinater

Dersom bildet som skal vises på skjermen inneholder objekter som er gjennomsiktige, skygger, spesielle lysforhold osv. kan renderen beregne hvordan disse enkeltkomponentene vil påvirke det endelige bildet hver for seg, og til slutt komponere det endelige bildet. Når beregningen deles opp slik, kalles hver beregning en renderpassering (render pass). [17]

2.1.3.3 Lys

En av motivasjonene bak det å gå fra 2D til 3D er for å gi grafikken mer dybdeføelse. For å få til en naturlig 3D-illusjon er det svært viktig med lys. Som figur 2.4a er det vanskelig å se at objektet er en kube uten en naturlig lyssetting.

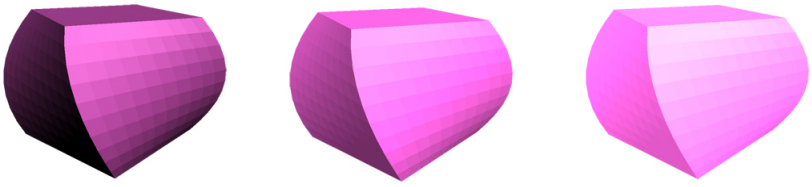
(a) Uten lys (b) Med lys

Figur 2.4: Viktigheten av lys i 3D-grafikk

I en 3D-Motor tegnes lys-verdier inn på bildet slik:

$$Polygonfarge \cdot Lysfaktor = pixelfarge$$

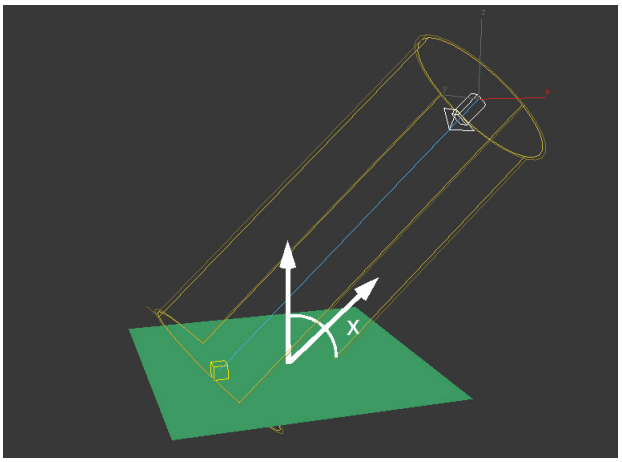
Lyskilder kan settes opp på forskjellige måter,direkte lys, og omgivelseslys. Omgivelseslys er en lysverdi som legges likt til alle flater på alle objekter i det 3d-dimensjonale rommet. Dette brukes hvis man ikke vil ha helt sorte områder på bildet, eller vil redusere graden av skyggelegging på objektene. I datagrafikk kan slikt lys brukes for å få lys i et rom uten nødvendigvis å ha en åpenbar lyskilde. I tillegg til omgivelseslyset som programmeren legger inn spesifikt, kommer mengden av lys fra alle direkte lyskilder i scenen.



Figur 2.5: Objekt påvirket av forskjellige mengder omgivelseslys

Direkte lys kommer fra lyskilder som er plassert i [X,Y,Z]. Disse kan deles inn i 3 typer, rettet lys, punktlys og spotlight.

Rettet lys kan sees på som lys som kommer fra en fjern kilde, for eksempel solen. Fordi kilden er fjern, kan lysstrålene sees på som parallelle. Dette er en fordel i beregning av scenen fordi man slipper å endre vinkel når man skal beregne lysets virkning.Rettet lys beregnes som et punkt med en retning i [X,Y,Z] altså en helt vanlig vektor. I tillegg legger vi inn ekstra verdier som lysfaktor og lysfarge



Figur 2.6: Rettet lys mot en flate

Punktlys er lys som kommer fra et bestemt punkt i scenen. Med dette lyset vil hvert punkt på et objekt få forskjellig intensitet, fordi det treffer med en litt annen vinkel. Dette lyset stråler ut i alle retninger i [X,Y,Z]. Dette vil treffe alle polygoner i scenen fra den vinkelen lyset befinner seg i forhold til polygonene. Det er såkalt omni-direksjonelt. Intensiteten, eller belysningen på polygonet blir beregnet utifra vinkelen lyset treffer overflaten. Ved å se på vinkelet til lyset i forhold til vinkelen på polygonets normal, dvs jo mer parallell med normalen, jo mer lys.

Spotlight er et punktlys der lyset er avgrenset av vinkler. Lyset kommer fra denne kilden som en kjegle. For å få dette lyset til å se mer realistisk ut, kan det også komme som to kjegler utenpå hverandre, der den innerste har et jevnt lys, mens i den ytterste blir lyset gradvis svakere mot kanten.

Objekter i scenen har en antributt ved seg som kalles material. Dette skal fortelle rendreren hvordan lyset skal oppføre seg når det treffer objektet. Materialer kan for eksempel være skinnende, matte, gjennomsiktige osv. [16]

2.1.3.4 Skygger

En annen måte å skape en god illusjon av 3D er å la objektene i scenen kaste skygger. En enkel måte å få dette til er å legge en tekstur av en skygge under objektet. Dette er gir imidlertid ikke et veldig realistisk resultat. Skygger kan enten være skarpe eller

utydelige. Skarpe skygger vil si at kantene på skyggen er skarpe, mens utydelige skygger har uskarpe kanter. Det finnes flere metoder for å generere skyggene i sanntid.[17]

Skygger på flater

Hvis skyggene kun skal vises på en flate kan man benytte seg av skyggeprojeksjon. Her blir punktene i objektet transfomert ned på flaten. Dette gir en del problemer. Projeksjonen kan ende utenfor flaten, man må passe på at projeksjonen ikke blir tegnet under eller for langt over flaten. Denne teknikken gir harde skygger. Hvis man ønsker utydelige skygger, må man ta større hensyn til lysets plassering i forhold til objektet. Se [17] for en nøyere utdyping av dette.

Skyggetekstur

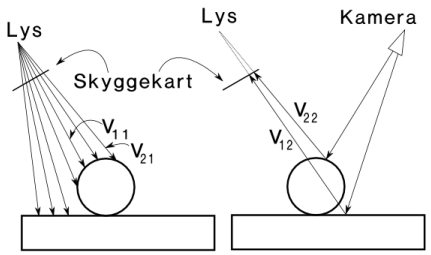
Hvis man ønsker å la skyggen falle på en ikke-plan overflate, kan man benytte seg av skyggetekstur. Denne metoden fungerer ved at silluetten av det som skal kaste skygge blir funnet, og denne blir lagt som en tekstur på den aktuelle overflaten. Her må både siluetten og kartleggingen mot objektet genereres, så metoden egner seg best på objekter som ikke endrer form.

Skyggevolum

Hvis man tenker seg linjer som strekker seg fra lyskilden og til hvert punkt på en figur, så vil skyggevolumet være det uendelige volumet som dannes av linjene som fortsetter etter de har truffet hvert punkt. Alt som ligger innenfor dette volumet vil være i skygge.

Skyggekart

Denne metoden lager først en tabell, kalt et skyggekart, over avstandene fra en lyskilde til alle punkter i scenen, se venstre side av figur 2.7 Deretter beregner renderen scenen på nytt, med synsvinkel fra kameraet. Avstanden mellom posisjonen til hvert punkt og lyskilden blir så sammenlignet med tilsvarende posisjon i skyggekartet, se høyre side av figur 2.7. Dersom avstanden er omtrentlig lik, så er punktet belyst. Er avstanden fra lyset til punktet betydelig kortere enn avstanden fra kameraet, så ligger punktet i skygge.



Figur 2.7: Generering av skyggekart, hentet fra [17], side 270

2.1.3.5 Teksturer

En tekstur er et 2-dimensjonalt bilde. Teksturer brukes for å få enkle 3D-dimensjonale objekter til å se mer komplekse ut. For eksempel kan man lage en mursteinsvegg ved å ha en enkelt flate, dvs 4 punkter, og legge et bilde av en mursteinsvegg på denne. Alternativet ville vært å lage hver enkelt murstein, og selv dette ville ikke blitt like realistisk, da hver murstein kan være forskjellig og ha mange små detaljer.

I en scene kan det være flere objekter som bruker samme tekstur, eventuelt kan et objekt inneholde mange mindre deler som alle bruker den samme teksturen. For eksempel kan man ha et tre, der alle blader skal bruke samme teksturen. For å ungå å måtte laste inn den samme teksturen om og om igjen kan det benyttes noe som kalles en teksturmanager. Denne holder oversikt over alle teksturer i scenen i en tabell, slik at teksturer som allerede er lastet kan hentes fra denne tabellen.
Ved å bare legge på et bilde på et objekt, oppstår det et problem. Enkelte deler av bildet viser kanskje metall, andre betong. Disse to materialene oppfører seg forskjellig når lys faller på dem. Dette kan løses ved å legge på nok en tekstur. Den nye teksturen forteller hvordan lyset skal reflekteres i hvert punkt. Tilsvarende kan man legge på teksturer med informasjon om små ugjenvnheter (bumpmapping), teksturer som fungerer som lys på en overflate (lysmapping) m.m.[17]

2.1.3.6 Shadere

En stor fremgang i prosesseringen av 3D-grafikk var da hardware for å utføre transformasjon og lyssetting (T&L) ble lagt til på skjermkortene. Dette gjorde at prosessen kunne optimaliseres, og dermed utføres raskere. Programmeren kunne sende inn parametre for å fortelle hvordan den ønsket opptegning av de forskjellige delene av scenen. Ulempen med denne metoden var at dersom man ønsket spesielle effekter, eller andre spesialiserte løsninger, kunne dette ikke gjøres på det hardkodede systemet. Etterhvert som skjermkortene ble kraftigere kom ønsket om å få direkte tilgang til GPUens instruksjoner for å lage egne funksjoner. Løsningen på dette kalles shadere. Shadere er små programmer, laget i et lavnivå programmeringspråk, som er ment å kjøre på GPUen. Shadere kjøres istedet for den tilsvarende prosessen på T&L-modulen. Det er med andre ord ikke mulig bare legge til funksjonalitet for prosesen. Hele regelsettet for rendringen må lages i funksjonen. De tre vanligste shaderne er vertexshader, geometrishader og pixelshader. [34] [16]

Vertexshader

Vertexshadere tar over jobben fra T&L-motoren. Punktene i en modell sendes inn til shaderen sammen med farge, tekstur-kooridinater og lignende. Shaderen transformerer så punktene til 2D-koordinater og en dybde-verdi. Ved å gjøre det mulig å programmere denne delen av rendreren kan man få til effekter som deformasjoner av bildet, feks. fiskeøye-linse, bølgeeffekter og lignende. Det er også mulig å legge animasjon inn til shaderen, for eksempel kan man få et flagg til å vaie i vinden ved å la vertexene følge en sinuskurve.[3]

Geometrishader

Geometrishaderen har muligheten til å legge til og fjerne punkter for å lage mer detajler i en figur. [11] Den tar inn et simpelt primitiv, det vil si et punkt, en linje eller et triangel, og sender ut en eller fler av en fastsatt type simpelt primitiv. Dette kan for eksempel brukes til å generere skygger, bevegelsessløring (motion blur), partikkelsystemer m.m.[13] En annen bruksmåte er å justere antall pixler i en figur avhengig av avstanden til kameraet. [19]

Pixelshader

Pixelshaderen finner fargen til hver pixel som skal vises på skjermen. Disse shaderne vet bare om èn pixel, og kan derfor ikke gjøre så kompliserte beregninger. Pixelshaderen bruker teksturer, lys, farge og andre attributter til å beregne pixelets farge. Et eksempel på hvordan det kan brukes er bumpmapping, der en av teksturene som brukes er et høydekart som viser større eller mindre fordypninger på en gjenstand. Pixelshaderen bruker denne informasjonen og endrer lysmengden som påvirker dette pixelet i samsvar med høyden gitt utifra høydekartet. [34]

2.1.3.7 Animasjon

Teknikkene for å animere karakterer i dataspill har endret seg opp igjennom årene. I spill som Doom og Wolfenstein3D ble det benyttet såkalte sprites. Sprites er ferdigrendrede 2-dimensjonale bilder, som i en 3-dimensjonal verden alltid vil være rettet mot spilleren. Ved å lage en rekke bilder som tilsammen utgjør en gå-sekvens, kan man skape illusjon av en karakter som beveger seg. For å gjøre illusjonen mer troverdig må man lage bilder av karakteren fra flere vinkler. En annen fremgangsmåte som gir mer realistiske animasjoner er 3D-modeller med nøkkelrammer. Her blir karakterene modellert i 3D, og deretter blir de satt opp i forskjellige positurer. Hver av disse positurene blir kalt nøkkelrammer. En ønsket animasjon vil da være å endre modellen fra en positur til en annen. Alle mellomliggende positurer beregnes ved interpolasjon. Denne teknikken er enkel å bruke, men filene som inneholder informasjonen om karakterene blir store da alle positurer må lagres som en modell. Det er også vanskelig å la karakteren reagere på fysikk-motoren, da alle animasjoner er ferdig lagret. Den fremgangsmåten som benyttes på nye spill i dag kalles skjelett-animasjon. Her blir en modell bundet opp mot et skjellet, der hvert bein i skjellettet kontrollerer en bit av modellen. Beinene er satt sammen av ledd, som kan påvirke hverandre. For eksempel vil en overarm som roterer også flytte på underarm og hånd.For å lage en animasjon trenger man dermed bare å lagre nøkkelrammene til skjellettet for hver positur. En gå-sekvens kan lagres som en rekke rotasjoner av ledd til bestemte tidspunkter. Det er heller ikke så vanskelig å la karakteren bli påvirket av fysikkmotoren, da hvert enkelt ledd kan påvirkes av krefter. Fordi man kan styre hvert enkelt ledd er det krevende å implementere denne metoden. [16]

2.1.3.8 Spesielle effekter

3D-motorer har ofte et utvalg av ferdig innebygde effekter som ofte er etterspurt. Eksempler på dette kan være flammer, tåke, solrefleks i kameralinse, vann, eksplosjoner, partikkelsystemer og lignede. Spill har ofte bruk for slike effekter, og det kan være en stor besparelse å slippe å programmere disse selv. Effektene bør være fleksible, slik at man lett kan tilpasse dem til sitt behov. Et partikkelsystem er et system av en mengde elementer med like egenskaper. Dette kan for eksempel være gnister fra et bål, der hver enkelt gnist er en partikkel. Partikkelsystemet settes opp ved at det legger til nye elementer i startposisjoner, som så blir påvirket etter bestemte regler. I eksempelet med gnister fra et bål kan nye gnister bli lagt til på tilfeldige steder i bålet, få en initiell fart oppover, for så å bli påvirket av gravitasjon, og etter en gitt tid, som også kan bli satt tilfeldig, for så å bli tatt ut av systemet og satt inn i startposisjon igjen.

Når man lager spillets omgivelser, kan det fort bli nødvendig med en eller annen slags himmel. For å få dette til å se naturlig ut kan man bruke en teknikk som kalles himmelboks.

2.1.3.9 Lyd og Video

For å skape stemning og for å lage en mer virkelig spillopplevelse er musikk og lydeffekter viktig. Det er derfor gunstig om spillmotoren har en enkel fremgangsmåte for å legge på dette. Det finnes i dag en rekke lydformater, feks wav,mp3,ogg,flac. Det kan være en fordel at spillmotoren kan importere hvertfall de vanligste filtypene.
Når man spiller av lyder i spillet kan de enten komme fra et bestemt sted, eller være såkalt ambient, dvs. lik over alt. For en lyd som kommer fra et bestemt sted må man justere volumet avhengig av hvor langt unna spilleren er, og hvilken retning han peker i. Her er det også mulig å gi ha forskjellig volum i de forskjellige høytalerne for å simulere hvilken retning lyden kommer fra.[16]
Det kan i spill også ofte være ønskelig å vise en video, enten som en scene mellom to spillesekvenser, eller som en del av spillomgivelsene. Som for lyden kan det da være en fordel å kunne laste inn mange forskjellige videoformater.

2.1.3.10 Nettverk

Dersom spillet man utvikler skal tillate at spillere skal kunne spille med/mot hverandre over et nettverk, må man legge inn funksjonalitet for dette. I dag er slik funksjonalitet ofte forventet i de fleste spill. Det kan derfor være en fordel om denne funksjonaliteten er innebygd i spillmotoren. Spill i nettverk kan enten settes opp som Peer-to-Peer, eller som tjener/klient. I et peer-to-peer oppsett snakker alle brukere med alle, og om en bruker mister forbindelsen har ikke dette noe å si for de andre spillerne. Denne metoden skaper mye datatrafikk, da all kommunikasjon må sendes til alle brukere. I et tjener/klient-oppsett fungerer en maskin (eller en maskinpark) som tjener, der spillerne kobler seg opp som klienter. Kommunikasjonen går bare mellom hver enkelt bruker og tjeneren. På denne måten kan mye av spillogikken kjøre på tjeneren, og dermed redusere hva klientmaskinen

må gjøre. På den andre siden, dersom tjeneren mister kontakt med nettverket, mister også alle klientene tilgangen til spillet. [16]

2.1.3.11 Fysikk-motor

Dersom spillet man utvikler tar for seg mye bevegelser og lignende, kan det være fornuftig å benytte seg av en fysikk-motor. Fysikkmotoren er en komponent som simulerer hvordan bevegelsene til alt i scenen utarter seg. Dette kan være alt fra kollisjonsdeteksjon, hvordan krefter påvirker objekter, fluidsimulering og til hvordan et klesplagg flagrer i vinden. Ved å bruke en fysikkmotor kan ting som en animatør ville brukt lang tid på å animere bli beregnet av datamaskinen. Fysikkmotorer har to hovedområder, rigide objekter og ikke rigide objekter. Rigide objekter kan ikke endre form, noe som gjør dem lettere å regne på. De utsettes for krefter som fører til translasjon og rotasjon. Ved å koble sammen rigide objekter med forskjellige ledd kan man lage komplekse systemer, for eksempel en modell av et menneske. Ledd kan gis restriksjoner på bevegelser for å få mer realistiske bevegelser. [7] Ikke-rigide objekter kalles også for "soft-body" objekter, noe som tilsier at de kan endre form. Dette innebefatter for eksempel fluider og klær. Disse får en rekke egenskaper som skiller dem fra rigide objekter: De har en mykhet, som kan justeres. Denne sier noe om hvor mye avstanden mellom punkter på objektet kan endres, og hvor stor kraft som kreves for å gjøre en endring. De har en elastisitet, som kan justeres. Denne sier noe om hvorvidt objektet går tilbake til opprinnelig form når kraften som påvirker den opphører, eller om den forblir i den nye formen. Dermed kan man eksempelvis simulere en sprettball, som får tilbake opprinnelig form, og leire, som vil få en ny form. Ikke-rigide objekter kan også kollidere med seg selv, og må dermed også gjøre disse beregningene.[14]

Fysikkmotorer i sin aller enkleste forstand kan være i et 2D-spill der spilleren ikke kan bevege seg innenfor enkelte områder på skjermen. Spilldesigneren har da på forhånd satt opp enkelte grenser der spilleren ikke får bevege seg forbi. Dette blir en svært enkel form for kollisjonsdeteksjon. Denne type kollisjonsdeteksjon gjør det mulig å sette begrensninger for hvor spilleren kan bevege seg. I dag forsøker spilldesignere å etterligne virkeligheten, slik at alle gjenstander som treffer andre gjenstander påvirker dem. Dette gjør det mulig å lage spill der man må bruke omgivelsene for å oppnå mål.

En av viktighetene med en fysikkmotor er at den bare gjør de beregninger som er nødvendige. I den virkelige verden er alle ting konstant i påvirkning av krefter. Skulle et komplekst spill ta hensyn til alle krefter på alle objekter ville den nødvendige regnekraften fort bli for stor. Det er derfor viktig at motoren kan velge ut hvilke beregninger som er nødvendige.

De fleste spillmotorer velger å benytte seg av ferdige fysikkmotorer isteden for å utvikle sine egne. I mange tilfeller har spillmotorene innebygget enkel kollisjonstesting, men for mer avansert fysikksimulering er det vanlig å velge et ferdig system. Noen aktuelle fysikkmotorer er:

2.1.3.12 Bullet

Bullet er en åpen kildekode-motor skrevet i c++. Den er laget under zLib-lisensen. Bullet har fysikksimulering for stive legemer som kan henge sammen med forskjellige typer. Den har også innebygd styring av kjøretøyer og rollefigurer. I tillegg har Bullet innebygget simulering av for myke legemer som klesstoff, tau og andre objekter som kan deformeres. [2]Dersom man ønsker å benytte seg av en motor skrevet i C++ i et javaprogram, må man lage et javagrensesnitt som benytter seg av denne motoren. Med denne fremgangsmåten må man kompilere c++ motoren en gang for hvert operativsystem man ønsker å benytte fysikkmotoren på.

2.1.3.13 jBullet

For å slippe å kompilere en c++ til hvert aktuellt operativsystem, kan man istedenfor oversette hele motoren til java, som kjører som på en virituell maskin og dermed kan kompileres en gang og kjøre på alle systemer. jBullet er en oversettelse til java fra den c++ baserte fysikkmotoren Bullet. jBullet er ikke en fullstendig oversettelse. Motoren har fysikksimulering for stive legemer, som kan kobles sammen med forskjellige slags ledd. Motoren kan brukes etter zLib-lisensen. [9]

2.1.3.14 ODE

ODE (Open Dynamics Engine) er en fysikkmotor skrevet i C++. ODE kan simulere bevegelse og kollisjon av stive legemer, som kan være festet med forskjellige slags ledd. Disse leddene kan også utstyres med en maksimal belastning, som gjør at de kan brytes. Massesenterene i objektene kan også plasseres der det ønskes. ODE har også simulering av friksjon mellom objekter. [20]

2.1.3.15 JOODE

JOODE står for Java Object Orientated Dynamics Engine er en oversettelse av c++ motoren ODE til java. Motoren er skrevet kun i java-kode, og kan derfor brukes på alle system som kan kjører java. Denne motoren er fremdeles i utvikling, så enkelte av egenskapene i ODE kan mangle i JOODE [21]

2.1.3.16 Verktøy

Enkelte spillmotorer er fullverdige programmer der spillene blir laget i et pek-og-klikk grensesnitt. De fleste spillmotorer har ofte en API i bunnen som lar brukeren benytte alle dens funksjoner. Mange av oppgavene som må utføres for å lage et spill kan være tungvinte å utføre som ren programmering. For eksempel er det unødvendig komplisert å bruke APIen direkte for å lage kompliserte 3D-modeller. Mange spillmotorer har derfor programmer der man kan utføre slike operasjoner på en bedre måte. Eksempler på verktøy er: 3D-modelleringsverktøy, tegneverktøy for terreng, spillebrett-redigerer, animasjonsredigerer osv.

2.1.3.17 Brukerinput

Et spill krever at brukeren på en eller annen måte kommer med input, enten det er gjennom tastatur, mus, joystick, akselerometere eller annet. Det er derfor fornuftig at spillmotoren har en ferdig modul som hjelper til med å motta slike input. Motoren bør både ha en enkel måte å ta i bruk brukerinput, men også gi muligheten til å endre og legge til funksjonalitet. I 3-dimensjonale spill kan man benytte piltastene for å styre rollefiguren, men mest vanlig er det i dag å benytte tastene WASD og musen. Her blir W brukt for å gå forover, S bakover, A til venstre og D til høyre. Disse tastene er blitt tatt i bruk fordi det er mer ergonomisk for venstrehånden å benytte taster på venstre side av tastaturet samtidig som man bruker høyre hånd til å styre musen.[16]

2.1.3.18 Informasjon til spilleren

For å gi brukeren informasjon om rollefigurens tilstand, benyttes ofte en HUD (Head up Display). Dette ordet kommer fra jagerfly, der informasjon om flyet kommer opp på en gjennomsiktig skjerm foran flygeren, så han slipper å se vekk fra vinduet for å få informasjon om flyet. En slik HUD kan i et spill kan også gi muligheten til å endre tilstander, utføre handlinger og lignende ved å være utstyrt med knapper. [30]

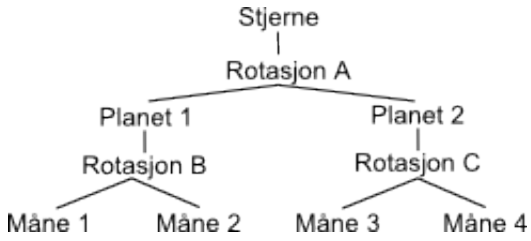
2.1.3.19 Kameravinkler

Et 3-dimensjonalt spill krever at scenene i spillet blir vist på en slik måte at det er mulig å oppfatte hva som skjer. De vanligste måtene å vise scener på er førstepersons synsvinkel, tredjepersons synsvinkel med følgekamera, og tredjepersons synsvinkel som ikke følger spilleren. Førstepersons synsvinkel gir inntrykk av at scenen blir sett gjennom spillerens egne øyne. Tredjepersons synsvinkel med følgekamera vil si at man kan se rollefiguren bevege seg i scenen, og bildet som vises er som fra et kamera som følger etter figuren. Den siste muligheten vil si at kameraet viser et oversiktsbilde av scenen. Kameraet er uavheng av en eventuell rollefigur, men kan enten være låst til en posisjon, eller kunne flyttes rundt for å se forskjellige deler av scenen.

2.1.3.20 Scenegraf

Når større mengder grafikk skal presenteres, blir det fort åpenbart at de forskjellige komponentene bør organiseres. Den vanligste måten å gjøre dette på er ved hjelp av en scenegraf. Scenegrafen er som navnet tilsier en graf, og vanligvis et n-tre. Objekter blir lagt til som noder i treet. Transformasjoner (feks. rotasjoner, skalering, translasjoner) kan legges inn som egne noder i treet, og vil dermed gjelde for alle barn av denne noden. Et hvert objekt som skal vises har en rotasjon og en translasjon. Når objektene er plassert i en scenegraf vil disse transformasjonene kunne finnes ved å traversere treet ned til den aktuelle noden, og multiplisere de aktuelle matrisene som ligger i treet. Når en scene skal tegnes traverseres hele treet, og matrisene beregnes etterhvert, og legges og hentes fra en stabel når treet deler seg.

Som et eksempel kan man se på animasjonen av et solsystem: (eksempelet er hentet fra [5])



Figur 2.8: Skjematisk oversikt over en scenegraf av et lite solsystem

Ved traversering får vi her:

Tegn stjernen
Lagre rotasjonsmatrisen
    Gang inn rotasjon A
    Tegn Planet 1
    Lagre nåværende rotasjonsmatrise
        Gang inn rotasjon B
        Tegn måne 1
        Tegn måne 2
    Hent den lagrede rotasjonsmatrisen
    Tegn Planet 2
    Lagre nåværende rotasjonsmatrise
        Gang inn rotasjon C
        Tegn måne 3
        Tegn måne 4
    Hent den lagrede rotasjonsmatrisen
Hent den lagrede rotasjonsmatrisen

En av fordelene ved denne representasjonen er at man slipper å beregne de totale matrisene for hvert objekt. Ved denne representasjonen kan man også lett rotere, flytte eller gjøre andre endringer på store deler av systemet ved å innføre en enkelt operasjon. Scenegrafen kan brukes også til å redusere beregningsmengden fordi man vet at når en node ikke synes, vil heller ikke barna til denne noden synes, og hele dette subtreet kan unnlates når scenen skal rendres. [5]

2.1.4 Hvilke Spill-motorer finnes

2.1.4.1 Underliggende motorer

Spillmotorer er komplekse programmer. Ofte er avanserte motorer bygget oppå mindre avanserte motorer. For eksempel kan en enkel motor som tilbyr rendering og håntering

av inngangsverdier fra bruker. En mer avansert motor kan tilby scenehåndtering som en scenegraf, fysikkhåndtering og lignende, men bruke den mindre avansete motoren som render. På det laveste nivået finner man det åpne systemet openGL og microsoft sitt Direct3D

Et par av de mindre avanserte motorene er JOGL, LWJGL og SDL. JOGL står for "'Java bindings for OpenGL"' og er et API som er startet av Sun Microsystems. LWJGL står for "'Light Weigth Java Game Library"'. Også dette baserer seg på openGl. SDL står for "'Simple DirectMedia Layer"' er et mye brukt c++ API. Også dette benytter seg av OpenGl.

2.1.4.2 Liste over åpen kildekode motorer

Det er mange som har forsøkt å lage spillmotorer. En del motorer er laget i forbindelse med utviklingen av et spill, og senere sluppet som åpen kildekode. Den følgende listen er laget med utgangspunkt i [32], og tar for seg en rekke spillmotorer som er mulig å bruke uten vederlag.

Tabell 2.1: Spillmotorer

Navn	Adresse	Språk	Lisens	Dim.	Type	Oppdatert
Agar	http://libagar.org/	C	BSD	2D/3D	Eget GUI	2008
Adventure Game Studio	http://www.adventuregamestudio.co.uk/	C++	Freeware	2D	Eget GUI	2008
Allegro Library	liballeg.org	C	gift-ware	2D	API	2008
Ardor3D	http://ardor3d.com	java	zLib	3D	API	2009
Box 2D	http://www.box2d.org/	C++	zLib	2D	API	2008
ClanLib	http://www.clanlib.org/	C++	BSD	2D	API	2008
Crystal Space	www.crystalspace3d.org	C++	LGPL	2D/3D	SDK	2008
Cube	http://www.cubeengine.com/	C++	zLib	pseudo-3D	Eget GUI	2005
DarkPlaces	http://icculus.org/twilight/darkplaces/	C	GPL	3D	Eget GUI	2008
Delta3D	http://www.delta3d.org/	C++	LGPL	3D	API	2008
DX Framework	http://dxframework.org/	C++	BSD	2D/3D	API	2007
Exult	http://exult.sourceforge.net/	C++	GPL	2D	Eget GUI	2004
Game Blender	http://www.gameblender.org	C++	zLib	3D	Eget GUI	2009
Genesis Device	http://www.GenesisDevice.net/	Object Pascal	LGPL	3D	API	2009
Irrlicht Engine	http://irrlicht.sourceforge.net/	C++	zLib	2D/3D	API	2008
Java 3D	https://java3d.dev.java.net/	java	GPL	3D	API	2009
jMonkey Engine	http://www.jmonkeyengine.com/	Java	BSD	3D	API	2009
Jogre	http://jogre.sourceforge.net/	Java	GNU	2D	API	2008

Fortsatt på neste side...

Tabell 2.1 – Fortsatt

Navn	Adresse	Språk	Lisens	Dim.	Type	Oppdatert
jPCT	http://www.jpct.net/	java	Egen	3D	API	2009
OGRE	http://www.ogre3d.org/	C++	LGPL	3D	API	2009
Open Scene Graph	http://www.openscenegraph.org/	C++	LGPL	3D	API	2009
ORX	http://orx-project.org/	C	LGPL	2D	API	2008
Panda3D	http://panda3d.org/	C++	BSD	3D	API	2008
PLIB	http://plib.sourceforge.net/	C++	LGPL	2D/3D	API	2005
Retribution Engine	http://www.apgardner.karoo.net/retrib/	C++	GPL	3D	Eget GUI	2008
Sauerbraten	http://www.sauerbraten.org/	C++	zLib	3D	API	2008
Simple DirectMedia Layer	http://www.libsdl.org/	C	LGPL	2D/3D	API	2007
Spring project	http://spring.clan-sy.com/	C++	GPL	3D	Eget GUI	2009
Stratagus	http://stratagus.sourceforge.net/	C++	GPL	2D	Eget GUI	2007
Troll 2D	http://code.google.com/p/troll2d/	c++	BSD	2D	API	2008
Verge	http://verge-rpg.com/	C++	BSD	2D	API	2007
Xilon Engine II	http://xilonengine.net/	Visual Basic .NET.	MIT	2D	API	2008
Xith3D	http://xith.org/	java	BSD	3D	API	2008

2.2 3D-modellering: Blender

Når man lager et spill kan man ofte i spillmotoren lage en rekke primitive figurer som bokser, kuler, kjegler osv. Disse kan igjen settes sammen til mer komplekse objekter. Dette er imidlertid en tidkrevende prosess der man heller ikke får øyeblikkelig tilbakemeldig om det man tegner ser riktig ut. Fremgangsmåten fungerer så lenge det er snakk om helt enkle objekter. Skal man tegne et mer komplekst objekt, er det fornuftig å ta i bruk en eget tegneprogram. Et slikt program, som er lansert som åpen kildekode, er Blender 3D. Blender er et kraftig 3D-grafikkprogram, der man blant mange andre ting kan modellere figurer. Figurene kan også få lagt inn parametre som tilsier hva slags materiale de er laget av. Dermed kan for eksempel en kule laget av glass reflektere lys forskjellig enn en kule laget av gummi. Objektene kan også få lagt på seg teksturer. Ved bruk av det som kalles UV-kart (se Appendix A.3) kan objektet brettes ut slik at flatene på figuren kan legges ned på et 2-dimensjonalt bilde. På denne måten kan et enkelt bilde gi teksturen til en komplisert figur. Når man har laget en figur i blender kan man eksportere denne i en rekke forskjellige formater, som deretter kan lastes inn i spillet.[24]

2.3 Metoder for å kjøre spill på nettsider

I oppgaven er det spesifisert at spillet skal kjøres som en java applet. Dette er imidlertid ikke den enste måten å kjøre et spill i en nettleser. I følgende avsnitt vil java applets bli

presentert, men også to andre metoder for å kjøre spill i nettleser vil bli nevnt.

2.3.1 Java Applets

2.3.1.1 Hva er en Java Applet

En applet er en et lite program, designet for å kjøre inne i et annet program. Java applets er applets som hovedsaklig er laget for å kjøre i nettlesere. Ved kompilering av programmet lages Java Byte Code, som kan kjøres i en Java Viritual Machine (JVM). For å kjøre appletten må man enten ha et tilleggsprogram i nettleseren, eller benytte seg av AppletVieweren til Sun Microsystems, som kan kjøre appletter direkte.

For å laste en applet i en nettleser må det først lages en HTML-side som peker på den gjennom en object-tag. Når denne HTML-siden lastes i nettleseren vil appletten bli lastet ned fra stedet den peker på, og deretter kjøres fra brukerens maskin. Programmet kjøres altså på brukerens maskin, og ikke på serveren. Programmet vil også normalt bli liggende i nettleserens mellomlager, og kan derfor startes raskere neste gang det skal kjøres. En ulempe med java applets er at de krever en plugin i nettleseren, og at JVM er installert på maskinen. Dersom brukeren sitter på en datamaskin som krever administratorrettigheter for å legge inn programvare, kan den ikke kjøres uten hjelp fra administrator.

2.3.1.2 Begrensninger ved Applets

Java Applets er avhengig av en plugin. Hvor mye minne hver java applett kan bruke er en instilling for denne pluginen. Dersom brukeren har satt denne grensen lavere enn utviklerern hadde regnet med, vil ikke appletten få nok minne. Appletten kjører i en såkalt sandkasse-modus. Dette vil si at den ikke har tilgang til noe på datamaskinen utenfor seg selv. I praksis vil dette si at den ikke får lov til å skrive eller lese fra filer på systemet. Det kan også være kompabilitetsproblemer, det vil si at dersom pluginen og appletten er av forskjellig java-versjon, kan det hende at programmet ikke fungerer. Hvis appleten er av en viss størrelse er det naturlig å samle filene i en bibliotekfil kalt en jar fil. For å kjøre en applet fra en nettleser må man signere appleten. Å signere appleten vil si å fortelle brukeren av appleten at programmereren går god for den. Programmeren kan kjøpe en signatur som beviser at det er han som har laget programmet. Det er også mulig å benytte seg av test-signaturer, men disse er bare gyldige i et halvt år om gangen, og gir ikke noen sikkerhet om hvem som faktisk har laget programmet. [15]

2.3.1.3 Hvordan lage en Applet

Det å komme i gang med å programmere en Java Applet er en enkel prosess. Java applets må utvide klassen "Applets", som inneholder metoder for å kunne vises i nettleseren. Et "hei verden"-program vil som en java applet se slik ut:

```
import javax.swing.JApplet;
import java.awt.Graphics;
```

```
public class HelloWorld extends JApplet {
    public void paint(Graphics g) {
 g.drawRect(0, 0,
    getSize().width - 1,
    getSize().height - 1);
        g.drawString("Hei verden!", 5, 15);
    }
}
```

2.3.2 Java Webstart

Java Applets har en del begrensninger, blant annet med minneforbruk og tilgang til lokale ressurser. En ny metode for å enkelt kunne kjøre applikasjoner fra nettleseren er Java Web Start. Java Web Start er programmer som startes ved å klikke på en lenke i nettleseren. Ulempen i forhold til applets er at de ikke så lett kan kommunisere med nettsiden de ble startet fra. Fordelen er at de er tilnærmet fullverdige applikasjoner som kan startes fra en nettside. [31]

2.3.3 Flash

Den mest populære måten å kjøre spill fra nettleser i dag er som adobe flash. Flash gir muligheten til å vise og lar brukeren påvirke animasjon på nettsider. Flash utvikles i et eget redigeringsprogram, og benytter et programerings-scriptspråk som heter action-script. Flash har imidlertid begrensede 3D-egenskaper, og er derfor ikke så aktuellt for denne oppgaven. [25]

Kapittel 3

Valg av spillmotor og design av testsystem

En av hovedhensiktene ved denne oppgaven var å velge ut en best mulig egnet spillmotor for Cyberlabs. Spillmotoren skal kunne kjøres sammen med de eksisterende modellsystemene deres. Disse modellsystemene er skrevet i Java, og det er derfor en fordel om den aktuelle spillmotoren også er skrevet i dette programmeringspråket.

3.1 Sammenligning av 3D-motorer

Tabell 3.1: Aktuelle spillmotorer

Navn	**Grafikk API**	**Objekt-organisering**	**Fysikkmotor**	**Verktøy**	**Applet**	**Lisens**
java 3D	Direct3D eller OpenGL	Scenegraf	Nei	Flere forskjellige	Ja	BSD
jPCT	LWJG eller JOGL	World-graf	Kollisjons-deteksjon	3. parts	Ja	Egen
xith3D	LWJG eller JOGL	Scenegraf	JOODE	N/A	spillet legges som panel	BSD
jIrr/Irrlicht	Direct3D eller OpenGL	Scenegraf	Innebygd eller ekstern	Ja	Nei	zlib/libpng
jMonkey Engine	LWJG eller JOGL	Scenegraf	JmePhysics	Monkey World 3D	Ja	BSD

Ut fra tabell 2.1 som presenterer 33 ulike spillmotorer har jeg vagt ut 5 motorer som kan være egnet for Cyberlabs. Disse er vist i tabell 3.1. Disse motorene velges på grunn av at de alle kan brukes med programmeingsspråket java. I tillegg har brukerbasen deres en viss størrelse. Dette er viktig fordi det da kan være gode muligheter for å få hjelp og svar på spørsmål fra andre brukere av motoren.
De ulike egenskapene til spillmotorerer i 2.1.3 er alle viktige, men de løses ofte på forskjellige måter. Disse må også tas hensyn til i valget av motor. Motorens ulike lisenser kan også ha en del å si for valget. Som nevnt i 2.1.2 kan det være vanskelig å finne ut hva slags egenskaper åpen kildekode-motorer har, da eneste tilgangen til informasjon om dem

kan være wikier som skjelden oppdateres. I det følgende avsnittet presenteres motorene i tabell 3.1. Spillmotoren jMonkey Engine blir lagt særlig vekt på, da denne er særlig interessant.

3.1.1 Presentasjon av Java 3D

Java 3D er en 3D motor som både brukes til visualisering og spill. Den er laget av Sun Microsystems. Informasjonen om Java 3D er hentet fra[23] og [12].

3.1.1.1 Sceneorganisering

Java 3D benytter seg av en scenegraf for å holde orden på objektene i scenen. Alle objekter i scenen er instanser av superklassen SceneGraphObject. Disse objektene er enten noder eller nodekomponenter. Noder er enten gruppenoder eller løvnoder (noder uten barn). Hver gruppenode i grafen kan ha en foreldrenode, og så mange barnenoder som trengs. Alle objekter i scenen er nodekomponenter, som blir referer av løvnodene. Nodekomponentene inneholder alt av informasjon om 3d-objekt, teksturer, materiale osv.

3.1.1.2 Lys, skygge og texturer

I Java3D er alt lys underklasser av løvnoder. Lys kan enten være omgivelseslys, punktlys, spotlight eller retningslys. Alle objekter kan ha forskjellige materialegenskaper som avgjør hvordan lyset reflekteres fra dem. Dessuten kan man velge forskjellige måter å tegne opp overflater, enten ved å la alle kanter vises, eller la objektet bli avrundet. Java 3D har bare enkle metoder for å tegne opp skygger. Motoren har en del avanserte metoder for bruk av textures, men har ikke vært i forkant med å implementere nye teknikker.

3.1.1.3 Fysikk

Java 3D har ikke noen intern fysikkmotor, men kan brukes sammen med en fysikkmotor.

3.1.1.4 Styring av rollefigurer

Java 3D har ikke ferdigbygde systemer for å kontrollere rollefigurer.

3.1.1.5 Forum

Java 3D har et forum, men brukere av motoren bruker ofte like gjerne andre forum for å spørre om hjelp. Motoren er imidlertid godt dokumetert, og det finnes flere bøker som tar for seg motoren.

3.1.1.6 Lisens

Man kan benytte seg av Java 3D motoren etter BSD-lisensen.

3.1.2 Presentasjon av jPCT

jPCT er en motor som faller litt utenfor oppgaven, i og med at den ikke er åpen kildekode. Den kan allikevel være aktuell for cyberlabs, i og med at den kan brukes uten vederlag. Informasjonen i dette avsnittet er hentet fra [4] og wikien på denne hjemmesiden.

3.1.2.1 Sceneorganisering

jPCT er bygget opp rundt et objekt som kalles World. Til hver world hører et kamera, lys, portaler og alle objekter som skal rendres i scenen. Hvert objekt har en rekke attributter. Av atrituttene må objektet inneholde minst en tekstur, transformasjonsmatriser (translasjon, rotasjon og skalering)og vektorer som tar for seg selve 3D-objektet osv. Objektet kan også inneholde animasjon, andre objekter som barn og et octtre.

3.1.2.2 Lys, skygge og texturer

På nettsidene til jPCT er inkonsistente med wikien deres anngående antall lys som kan påvirke hvert objekt. Nettsiden sier at ubegrenset antall lys kan påvirke hvert objekt, mens wikien mener det er kun de 8 viktigste lyskildene. Dette har mest sansynlig ikke betydning for dette prosjektet. Ellers har motoren støtte for de vanlige lystypene.
Motoren har lagt til rette for generering av skygger gjennom skyggekartlegging. Det finnes kun en TeksturManager som alle objektene bruker. Denne kjenner dermed til alle teksturene som brukes i programmet, og alle objekter kan dermed bruke alle teksturer. Hvis hardware-renderen brukes kan opptil 4 tekstur-lag legges på hvert objekt.

3.1.2.3 Fysikk

jPCT har 4 forskjellige kollisjonsdeteksjonsmetoder. Hvis man ikke spesifiserer en spesiell metode skjekkes de to kollisjonskandidatenes 3D-data mot hverandre. Dette kan gå utover ytelsen, derfor har jPCT 3 forenklede metoder. Den første, stråle mot polygon, egner seg til små objekter som kuler eller en laserstråle. Metoden beregner om objektet/strålen vil treffe polygonet i neste tidsskritt. Kule mot polygon metoden lager en kule rundt objektet som skal testes polygonet. Dersom kulen krysser polygonet vil objektet flyttes tilbake langs normalvektoren til polygonet. Denne metoden har en begrensning, da hvert polygon kun skjekkes en gang. Dermed kan skjekk på polygon nummer 2 sørge for at objektet blir flyttet inn i polygon 1. Ellipsoide mot polygon-metoden innhyller objektet i en ellipsoide. Metoden fungerer på samme måte som kule mot polygon, men her blir metoden kalt igjen når objektet blir flyttet tilbake, så man ikke beveger seg inn noen av polygonene.For mer avansert fysikk må man benytte en ekstern fysikkmotor.

3.1.2.4 Styring av rollefigurer

jPCT har ikke ferdige klasser for styring av rollefigurer. Man har tilgang til inngangsverdier fra tastatur, mus og eller joystick, og må lage kontrollene selv ut i fra dette.

3.1.2.5 Forum

jPCT har et forum på hjemmesidene sine med ca 400 registrerte brukere. Forumet er ikke veldig aktivt, men brukerne kan svare på spørsmål, og man kan finne eksempler på bruk av motoren. Aktiviteten kan sansynligvis ha noe med det å gjøre at koden ikke er helt åpen, og dermed begrenser brukerne fra å endre på den.

3.1.2.6 Lisens

jPCT er lansert under en lisens som ikke gir brukeren mulighet til å endre på koden uten tillatelse fra forfatteren av motoren. Man har lov til å dekomprimere filene for å legge de inn i sine egne jar-filer (eller lignende). Forfatteren krever ikke at man nevner at man bruker motoren, men ønsker gjerne å hvertfall få en referanse til motorens hjemmeside.

3.1.3 Presentasjon av xith3D

Xith3D er en en overbygging over en rekke subsystemer. Man kan da velge å la JOGL eller LWJGL være denne underliggende motoren. Andre systemer som benyttes er JAGaToo (Java Abstract Gaming Tools), JOPS (Java Open Particle System) osv. Informasjonen i dette avsnittet er hentet fra [6] og fra wikien på denne hjemmesiden.

3.1.3.1 Sceneorganisering

Xith 3D er bygget opp rundt scenegraf-arkitekturen. Noder som ikke er løvnoder i denne scenegrafen kalles grupper. De ulike delene av scenen blir festet til rotnoden med en node kalt BranchGroup. Til hver slik Branchgroup kan det kobles flere renderpasseringer. En BranchGroup kan så få koblet på seg en eller flere transformasjonsgrupper (TransformGroup), som utfører en transformasjon på alle underliggende noder. Hvis det ikke er nødvendig med en transformasjon kan man også bruke en ren gruppe (Group) for å holde på flere undernoder. Til disse transformasjonsgruppene eller gruppene kobles nye grupper eller løvnoder. Løvnoder som inneholder geometri kalles i Xith3D for Shape3D. Ved å ha flere BranchGroup kan man ha flere rendrepasseringer. På denne måten kan den siste renderpasseringen være for en HUD som skal ligge over alt annet.

3.1.3.2 Lys, skygge og texturer

Lyskilder blir i Xith3D lagt til i en gruppe i scenegrafen. Alle objekter som er i denne gruppen eller subgrupper av denne gruppen blir påvirket av lyset. Lysene kan være omgivelseslys, rettet lys, punktlys eller spotlight. De har støtte for en rekke bildeformater for textures. Det er også lagt til rette for flere multi-teksturer. Motoren kan vise skygger enten gjennom skyggevolumer eller skyggekart.

3.1.3.3 Fysikk

Motoren benytter seg av fysikkmotoren JOODE (Java Object Oriented Dynamics Engine). De har også lagt inn et partikkelsystem.

3.1.3.4 Styring av rollefigurer

Xith3D benytter seg av et styringssystem som heter JAGaToo. Man kan selvfølgelig få direkte tilgang til tastetrykk og musebevegelser, men motoren gir tilgang til flere ferdige klasser for hvordan inngangsverdier skal brukes.

3.1.3.5 Forum

Xith3D har et relativt aktivt forum med brukere som bruker motoren i sine egne prosjekter. Forumet blir brukt til å rapportere feil i motoren, og mange av brukerne er også med på å finne løsinger på disse feilene.

3.1.3.6 Lisens

Xith3D er lansert under BSD lisensen.

3.1.3.7 Spesiellt for Xith3d

Xith3D har sit eget HUD-system, kalt kalt Xith3D HUD (Heads-Up Display) som egner seg spesiellt for spill.

3.1.4 Presentasjon av jIrr/Irrlicht

Irrlicht er en mye brukt åpen kildekode-motor skrevet i C++. Motoren kan kjøres på svært mange systemer. Motoren er liten, og krever lite minne. Dessuten er den kompatibel med både ny og gammel hardware. Det er laget koblinger til flere andre programmeringsspråk. Koblingen mot java kalles Jirr. Det skal være mulig å kjøre jirr som en applet. Informasjonen i dette avsnittet er hentet fra [8], wikien og forumet på denne hjemmesiden, spesiellt tråden om jIrr på forumet.

3.1.4.1 Sceneorganisering

Irrlicht benytter seg av scenegraf for å organisere objektene i scenen. Rotnoden i scengrafen kalles SceneManager, og objekter legges til denne ved å koble dem på som noder på SceneManager. Nodene kan inneholde en transformasjon (translasjon, skalering, rotasjon). Nye noder kan legges på disse igjen ved å fortelle noden hva som er foreldrenoden deres.

3.1.4.2 Lys, skygge og texturer

Irrlicht har støtte for alle de vanlige lystypene, og har også mulighet for å benytte seg av anvanserte shadere. Motoren benytter seg av skyggevolum for å vise skygger. Motoren kan laste inn de fleste bildeformater for å bruke som textures. Den har også en rekke avanserte bruksområder for teksturer, som f.eks. animerte teksturer, bumpmapping osv.

3.1.4.3 Fysikk

Irrlicht har innebygget kollisjonsdeteksjon, men for mer avansert fysikksimulering kan man benytte seg av PAL (Physics Abstraction Layer), som kan kobles opp mot en rekke fysikkmotorer.

3.1.4.4 Forum

Brukerne på forumene til irrlicht er aktive, og det utvikles både utvidelser og verktøy for motoren. Fordi irrlicht er en av de mest populære spillmotorene basert på åpen kildekode, er det også mange som både rapporterer og retter feil i motoren. I og med at java er et populært språk for å programere applikasjoner til nett og mobiltelefoner er det også en del aktivitet rundt javakoblingen jIrr.

3.1.4.5 Lisens

Irrlicht og jIrr er lansert under zlib-lisensen, og kan derfor fint brukes til kommersielle produkter.

3.1.5 Presentasjon av jMonkey Engine

jMonkey Engine (JME) er startet av Mark Powell i et forsøk på å lage en god høynivå spillmotor for java. Den er i utgangspunktet basert på LWJGL, men kan også kjøres på JOGL. Informasjonen i dette avsnittet er hentet fra [18] og fra wikien på denne hjemmesiden.

3.1.5.1 Sceneorganisering

Scenegrafen i jMonkey er et tre bygget opp av Noder og Geometri. Begge disse er underklasser av klassen Spatial. Som navnet tilsier er dette deler av jMonkey som har med de "'romlige"' 3D-objektene i programmet. Nodene bygger opp skjelettet i grafen, mens Geometrien er løvnoder. En node kan med andre ord ha både noder og geometri som barn, mens geometri ikke kan ha barn.
Geometrien er objekter som inneholder informasjon om punktene som tilsammen utgjør 3D-objektene som skal tegnes. I tillegg ligger det informasjon om farge, hvordan teksturer skal legges på objektet, normaler som forteller hva som er utside og innside av objektet. Både Noder og Geometri kan i tillegg inneholde en rekke attributter.
Transformasjoner består av nodens plassering, rotasjon, skalering. Her kommer en av fordelene med scenegrafen frem, da en transformasjon på en node vil også gjøres på alle nodens barn. Her kan man både se for seg at et komplekst objekt med mange noder lett kan flyttes, roteres og skaleres ved å utføre transformasjon på den øverste noden i objektet. Et annet eksempel er hvis en 3D-modell av et menneske skal animeres, og man ønsker å rotere den ene armen, så vil både underarm, overarm, hender og fingre roteres riktig samtidig.
En annen attributt er BoundingVolume, som er et volum som inneholder noden og alle

dens barn. Dette volumet brukes for å finne ut om to objekter kolliderer. Dette gjøres ved å først undersøke om et aktuellt objekt er innenfor BoundingVolumet til den øverste noden i objektet som kollisjonstesten utføres på. Er den det må det skjekkes for kollisjon mot de lavere nodene. Ved å bruke scenegrafen plassere kollisjonstestene i et tre på denne måten kan man redusere antall nødvendige kollisjonstester.
Renderstate er en antributt som sier noe om hvordan geometrien skal tegnes opp. Dette kan for eksempel være hvilken tekstur som skal brukes, materialegenskaper, lys osv. Også denne attributten arves nedover til nodenes barn. På denne måten kan grupper av objekter som skal ha de samme teksturene ha same foreldrenode, og man slipper å endre renderstate for hvert objekt, noe som sparer prossessortid.
I tillegg kan Noder og Geometri inneholde en kontroller som beskriver animasjon og bevegelse for geometrien. Selve kontrolleren blir ikke arvet av barna av nodene her, men eventuelle transformasjoner som skjer som konsekvens blir arvet på vanlig måte.

3.1.5.2 Kjøring av et program

jMonkey har en rekke klasser man kan benytte for å komme raskt i gang med å lage et spill, for eksempel SimpleGame og StandardGame. De forskjellige klassene gir mulighet til å kjøre spill der bildeoppdateringen er mest mulig konstant, gi en makstid til beregninger for hvert oppdatering av bilde, osv. Noen av disse klassene har også ferdig oppsatte lys, kameraer osv for å gjøre oppstart av spillprogrammeringen enkel. Et spill starter med initialisering av lys, kamera, inngangshåndterere, objekter, scenegraf osv. De innebygde klassene gjør en rekke slike initaliseringer, og lar så programmereren kjøre sine initialiseringer med metoden simpleInitGame. Det er i denne metoden programmereren legger inn komponentene som skal være med i spillet. Når initialiseringen er ferdig går spillet inn i en løkke. Denne spilløkken sørger for oppdateringer av alle objekter og noder. For eksempel blir inngangshåndterer oppdatert med tastetrykk som har kommet inn. Deretter blir metoden simpleUpdate kjørt, som er en metode som programmereren selv kan fylle ut i koden sin. Her kan alle objekter programereren har lagt inn oppdateres som han selv vil. Når denne koden er kjørt, blir scenen sendt til renderen, som tegner opp scenen til skjermen. Deretter starter løkken på nytt igjen.

3.1.5.3 Lys, skygge og texturer

I jMonkey kan man sette opp punktlys, spotlight, retningsbestemtlys og omgivelseslys. Lys blir lagt til på samme måte som andre objekter i scenegrafen. Man har mulighet til å vise skygger ved å benytte seg av skyggevolum eller skyggekart. jMonkey benytter seg også av en teksturmanager, slik at en tekstur kan benyttes på flere objekter. Motoren har også mulighet til å benytte teksturer til bumpmapping, multitekstur osv. Alle objekter kan få satt et materiale som forteller renderen hvordan lys skal reflekteres fra dette objektet.

3.1.5.4 Fysikk

jMonkey har innebygd kollisjonsdeteksjon, og har dessuten noen enkle systemer for simulere klesstoff, vind og andre ting. De har også innebygget en partikkelgenerator. I tillegg har jMonkey en ekstramodul kalt jMonkey Physics. Dette er en overbygging over andre fysikkmotorer. Modulen har mulighet for å benytte seg av fysikkmotorene ODE, JOODE og PhysX. Av disse er det ODE som har den beste koblingen. jMonkey Physics gir jMonkey fysikksimulering av stive legmer med forskjellige type ledd (som også kan ha en maksstyrke, og dermed kan ryke).
For å lage et spill i jMonkey som benytter seg av fysikkmotoren, finnes det ferdige klasser man kan ta utgangspunkt i. Disse setter opp et fysikkrom"(PhysicsSpace). Ut fra dette fysikkrommet kan man lage statiske eller dynamiske noder. De statiske nodene vil beholde sin romlige posisjon, mens de dynamiske kan påvirkes av krefter. Når et dynamisk objekt blir generert blir det automatisk lagt til en konstant kraft nedover, for å simulere gravitasjon. Både statiske og dynamiske noder har en parameter kalt material, som gir den en rekke egenskaper, som for eksempel friksjon. Dynamiske noder har også en vektparameter som også er med på å bestemme hvor mye kraft som trengs for å flytte på det. Fysikknodene fungerer ellers som vanelige noder i jMonkey. Geometri kan kobles til dem, men en fysikknode kan ikke kobles på en annen fysikknode. Hvis to fysikknoder skal henge sammen, må man bruke et ledd. Fysikknodene kobles til scenegrafen på samme måte som andre noder.
Man har mulighet for å legge inn funksjoner som kjøres før og etter fysikkberegningene, hvis man har behov for dette.

3.1.5.5 Styring av rollefigurer

jMonkey har flere muligheter for å styre rollefigurer. Motoren tilbyr ferdige klasser som FirstPersonHandler, ThirdPersonHandler, NodeHandler og lignende for å kunne styre rollefigurer på forskjellige måter. I tillegg kan man selvfølgelig også ta imot tastetrykk, musebevelgelser og klikk og verdier fra joystick og benytte dem som man ønsker selv. Hvis man ønsker å sette opp sin egen håndterer, kan man utvide en klasse fra klassen InputHandler. I en slik klasse kan man så tillegne forskjellige taster forskjellige verdier, for eksempel kan piltast mot venstre få verdien "'venstre"'. Deretter kan man lage handlinger (actions) som skal utføres når håndtereren mottar denne verdien.
jMonkey har muligheten til å sette opp et følgekamera som kan følge etter rollefiguren. Dette har en rekke parametre for å fortelle programmet hvordan dette kameraet skal flytte seg.

3.1.5.6 Forum

JME har et meget aktivt forum som kommer med forslag til oppdateringer av motoren og svarer på spørsmål fra andre brukere. Hjemmesiden til motoren har også en wiki som blir oppdatert av brukerne.

3.1.5.7 Lisens

jMonkey er lansert under en modifisert BSD-lisens. Motoren kan brukes til kommersielle spill, men de ønsker at motoren på en eller annen måte blir nevnt.

3.1.5.8 Applets

I jMonkey finnes det en klasse, simleJMEApplet, som gir muligheten til å kjøre spillmotoren i en applet. Denne klassen er satt opp anneledes enn klassene man bruker for å lage et vanlig spill. Metodene som benyttes har andre navn, og andre metoder brukes for å få tak i renderen og annet.

3.1.5.9 Animasjon

Objekter kan flyttes rundt i scenen ved å la posisjonen deres endres for hver gjennomkjøring av spilløkken. Dersom man ønsker mer avansert animasjon, har jMonkey muligheten til å laste inn modeller som er satt opp med skjelett-animasjon.

3.1.5.10 Verktøy

Det er utviklet et verktøy kalt Monkey World 3d. Dette er et redigeringsprogram for scener og objekter i jMonkey. Verktøyet bygger på java-redigereren eclipse, og gir muligheten til å bygge opp terreng, plassere objekter osv.
For å lage komplekse objekter i jMonkey kan det være lurt å benytte et eksternt program, som for eksempel Blender 3d (se 2.2). jMonkey har innebygget funksjonalitet for å kunne laste inn en rekke 3d-modell formater, men beklageligvis ikke blenders standardformat. Blender kan imildlertid lagre filene sine som for eksemlpel obj-filer, som kan lastes inn av jMonkey.

3.2 Valg av spillmotor

Cyberlab sine nettspill er javaapplets, og deres kompetanse er fokusert på java. Det vil derfor være best om motoren som velges er skrevet i java. Alle motorene som er valgt ut som alternativer har gode egenskaper når det gjelder grafikk. De er alle lette å sette seg inn i. Med tanke på å ønske seg en javabasert motor, faller jIrr fort bort, i og med at det er en javagrensesnitt til c++ motoren irrlicht. jPCT er ikke åpen kildekode i ordets rette forstand, da koden ikke er tilgjengelig. Man kan gjennom lisensen dekompilere java-filene, og ved å spørre utviklerern om lov endre på koden. Dette er allikevel en begrensning som kan er lite ønskelig. Java 3D har fordelen av å være meget godt dokumentert. Denne motoren mangler en del egenskaper man ønsker seg ved en spillmotor, som god kobling mot fysikkmotor og enkel styring av rollefigurer. Denne motoren er også en mer generell motor, som også brukes til visualisering.
xith3D og JME har ganske like egenskaper. De er begge basert på den samme underliggende motoren. Xith3D har et bedre utviklet system for HUD. På den andre siden er

forumet til jMonkey mye mer aktivt. Dette er et viktig kriterie for valg av spillmotor. jMonkey har en del fysikksimulering innebygd i motoren, og har også sørget for en god kobling mot andre fysikkmotorer. Cyberlabs hadde på forhånd kommet med forslag om å benytte jMonkey Engine, men ønsket å vurdere alle aktuelle spillmotorer. Ut fra mine vurderinger har jeg kommet fram til at det er riktig å satse på denne samme spillmotoren. Fremgangsmåten for å laste ned og gjøre jMonkey Engine klar til bruk er beskrevet i vedlegg A.4.

3.3 Testprogram: nivåregulering i tank

For å teste ut om Cyberlabs modellsystem kan kjøres sammen med en spillmotor må det lages et testprogram. Hensikten med programmet er å se at spillmotoren kan brukes sammen med modellsystemet, samt å teste ut spillmotorens muligheter.
Cyberlabs forslag til testprogram er et tanksystem der en vanntank skal fylles av et innløp samtidig som vannet renner ut av tanken i et utløp. Både innløpet og utløpet skal kunne styres av ventiler. Ventilene skal både kunne styres manuellt og automatisk. Et panel utenfor selve spillet skal kunne stille på regulatorparametere for automatisk styring. Et annet panel skal kunne endre ventilåpningene manuelt.
En rollefigur skal kunne styres rundt tanken. Rollefiguren skal kunne manipulere størrelsen på tanken , og endre på ventilåpningene. Det skal ses på muligheter for å la rollefiguren endre regulatorparametrene.
Vanntanken skal kunne endre form, fra en kube til en kjegle. For å veksle mellom disse to kan det enten være et panel med valgmuligheter, lage en knapp rollefiguren kan manipulere eller begge deler.

3.4 Krav for systemet

Målet for denne oppgaven er å benytte en spillmotor for å utvikle brukergrensesnitt for en reguleringsmodell som skal kunne kjøre i en applet. Et selvfølgelig krav er da at systemet faktisk kjører som en applet i en nettleser. For å teste systemet videre, skal en rekke faktorer vurderes. Systemet testes på to maskiner.

Maskin 1

- Prosessor: Intel Pentium 4 3.00GHz
- Minne: 2Gb
- Skjermkort: Internt på hovedkortet
- Operativsystem: Windows XP

Maskin 2

- Prosessor: Intel Core 1,73GHz
- Minne: 1Gb
- Skjermkort: Internt på hovedkortet
- Operativsystem: Ubuntu 9.04

For å kunne vurdere systemet velger jeg å teste systemet etter følgende punkter:

Enkelhet - hvor lett er det å bruke motoren

Hovedpoenget med å benytte en spillmotor er at det skal være enklere og raskere å utvikle spill. Derfor er det også viktig at motoren har enkle løsninger for å få til det man ønsker å gjøre, samtidig som det har mange muligheter for å spesialisere spillet.

Grafikk

Når man benytter motoren er det også viktig av ting ser ut som forventet. Det skal vurderes om motoren viser objektene og scenen som man skulle forvente. For eksempel at alle teksturer blir vist, at alle grafiske effekter er som forventet osv.

Bildefrekvens kontra oppløsning

Når man skal benytte seg av 3D-grafikk er det viktig å huske på at det er mer krevende enn 2D-grafikk. Ikke bare på grunn av den ekstra dimensjonen, men også på grunn av lys-beregninger mm. Jo mindre oppløsning man har på skjermbildet som skal vises, jo færre punkter på skjermen må renderen beregne fargen av. Antall punkter på skjermen har derfor mye å si for hvor raskt et bilde kan oppdateres.Hvis bildefrekvensen kommer under et visst nivå, vil spilleren oppfatte hvert enkelt bilde som enkeltbilder, og ikke en gjevn strøm. Ved å justere oppløsningen på bildet vil man da kunne endre bildefrekvensen. Spillet må kunne ha en viss oppløsning for at det skal være mulig å bruke brukergrensesnittet på en fornuftig måte. Det skal derfor testes hvordan effekt forskjellige oppløsninger har på de to maskinene.

Kontrollering av spillet

Oppgaven skal utvikle et brukergrensesnitt mot en simulator. Det skal være mulig å endre på verdier i denne simulatoren, i tillegg til at man skal kunne bevege seg rundt i spillet. Dette må også være mulig når spillet kjøres fra en nettleser. Samtidig skal spillet illustrere hvordan en tank fylles og tømmes ved forskjellige åpninger på ventiler. Dette må komme frem på en god måte i spillet.

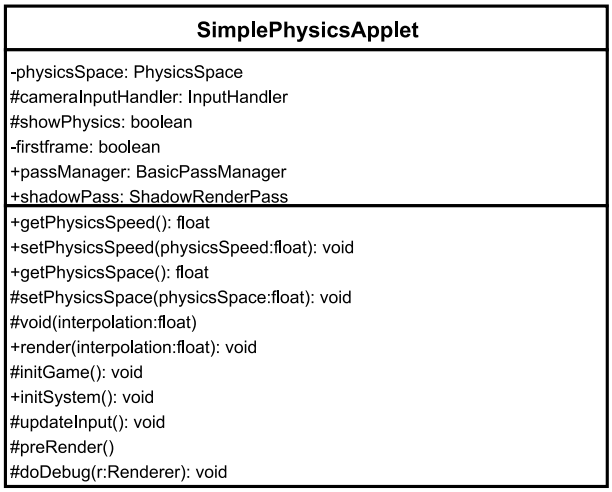
3.5 Design av det endelige systemet

En designprosess er ofte en iterativ prosess. Gjennom implementering og uttesting av et opprinnelig design, oppdages det uventede problemer og alternative løsninger. Denne oppgaven er ikke noe untak. Det at en del av oppgaven er å sette seg inn i spillmotoren impliserer at spillmotorens egenskaper ikke er helt kjente. En åpen kildekode-spillmotor som jMonkey kan også ha feilaktige beskrivelser av sitt eget system, og dermed gi mulighet til å ta designvalg som viser seg å feile.
I de følgende avsnittene presenteres designet for testsystemet. Designet presenteres med

klassediagram for hver klasse som skal programmeres for testsystemet. Systemet består av et tanksystem, en rollefigur som kan styres, og omgivelser. Simulatoren til Cyberlabs er tilgjengelig som en jar-fil kalt TankSimulatorWrapper. Denne har funksjoner som gir muligheten til å endre parametre og som hente ut systemets verdier. Det velges også å lage en klasse som skal ta seg av kall mot denne simulatoren. En viktig del av dette testspillet vil naturlig nok være grafikken. Det er vanlig å tegne grafikken i spill i egne programmer. Helt enkle figurer som bokser, kuler, kjegler osv. er lette å lage direkte fra API-en til spillmotoren. Som tegneprogram benyttes Blender 3D, se avsnitt 2.2. Før man går i gang med modelleringen av objektene er det en fordel å ha en ide om hvordan objektene skal se ut. Det er derfor også laget konsepttegninger av en del av gjenstandene i scenen.

3.5.1 Kobling mot jMonkey

jMonkey tilbyr en rekke klasser for å komme raskt i gang med å lage et spill, bla BaseGame og SimpleGame, se 3.1.5.2. For applets finnes kun en klasse, JMESimpleApplet. Under forsøk med å få testet denne, viser det seg at det er problemer med å bruke den i jMonkeys nåværende versjon. Brukerne på jMonkeys forum har kommet med en løsning der man kan lage applets på samme måte som andre spill [10]. Disse klassene fungerer godt, men som beskrevet i avsnitt 3.5.6.1 er det problemer med det innebygde kollisjonssystemet til jMonkey. For å løse dette valgte jeg å benytte meg av jMonkeys fysikkmotor. For å koble seg mot denne måtte jeg programmere en klasse som jeg velger å kalle SimplePhysicApplet.Jeg utvider klassen SimpleApplet fra forumets løsning [10] med koden fra SimplePhysicsGame fra fysikkmotoren. I tillegg legges det i denne klassen inn muligheter for å kunne la objektene kaste skygger Et klassediagram for klassen SimplePhysicsApplet er vist i figur 3.1.

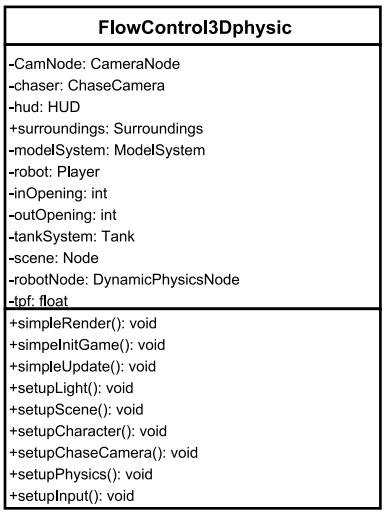


Figur 3.1: Klassediagram for SimplePhysicsApplet

Oppstart av spillet, og hva som skjer i spilløkken, tas hånd om i klassen FlowCon-

trol3DPhysic, se fig 3.2. For å få oversikt over programmeringskoden er det en fordel at filen der spilløkken er inneholder minst mulig kode. Denne klassen skal sørge for å starte alle andre deler av programmet, og oppdaterer alle objektene for hver gjennomgang av spilløkken. For å øke oversikten av programmet velges det å dele opp initialiseringen i deler som har noe med hverandre å gjøre. For eksempel legges alt som har med oppsett av lys i scenen i setupLight.

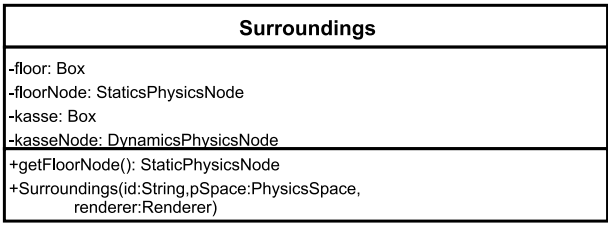
3.5.2 Oppstart og spilløkke



Figur 3.2: Klassediagram for komponentene

Omgivelser, se 3.3 er en klasse som skal inneholde alle gulv, vegger og andre objekter i scenen, bortsett fra tanksystemet. For å teste ut fysikkmotoren legges det inn en boks som rollefiguren kan kollidere med. Klassen tar inn fysikkrommet som spillet benytter, slik at fysikk-noder kan opprettes inne i klassen.

3.5.3 Omgivelser

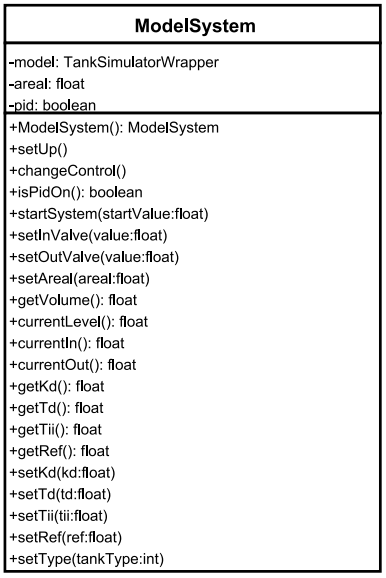


Figur 3.3: Klassediagram for klassen Surroundings

ModellSystem , se figur 3.4, er en klasse som benytter seg av Cyberlabs simulatorsystem som ligger i pakken TankSimulatorWrapper. Denne klassen har metoder som

gir tilgang til de forskjellige parametrene i simulatoren. I tillegg har denne klassen muligheten for å skru av og på pid-regulatoren i simulatoren, og velge type tank som skal brukes.

3.5.4 Modellsystem

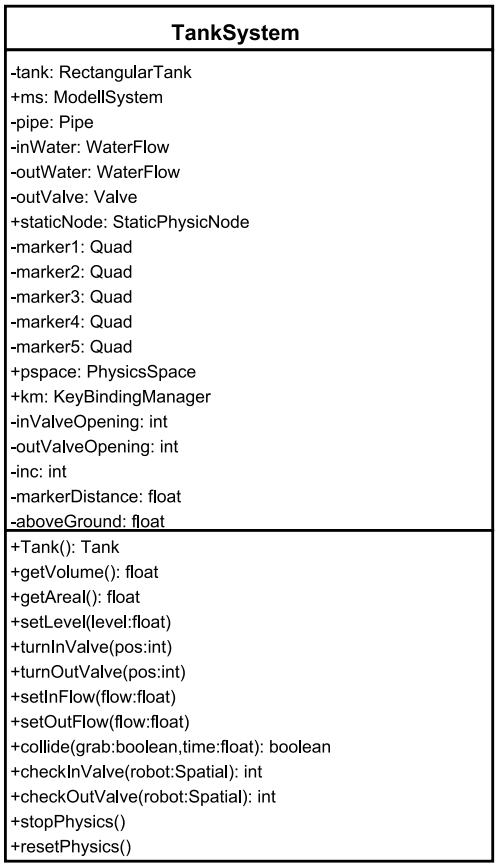


Figur 3.4: Klassediagram for klassen ModelSystem

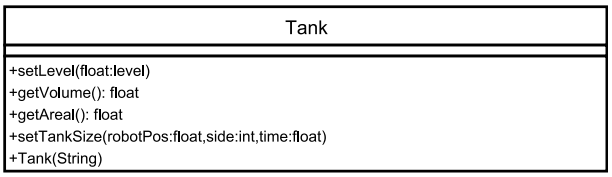
3.5.5 Tanksystem

Tanksystem er klassen som gir en grafisk fremstilling av matematikken i simulatoren. Tanksystemet vil bestå av en tank med vann, et rørsystem som kan fylle tanken, vannstråler inn og ut av tanken og ventilratt for å stille på vannmengden inn og ut av tanken. Dette gir fire parametre som må kunne styres, nivået i tanken,størrelsen på tanken, mengden vann som strømmer inn og mengden vann som strømmer ut av tanken. I Cyberlabs spesifikasjon bes det om at det skal være mulighet for å ha forskjellige type tanker. Denne funksonaliteten er imidlertid ikke lagt inn i simulatoren fra Cyberlabs. Av den grunn blir det lagt til rette for at man kan legge inn forskjellige former, men kun en rektangulær tank blir lagt inn i spillet.

For å legge til rette for at tanken kan ha forskjellig slags form, lages en abstrakt klasse , se fig 3.6 som de forskjellige tankene må arve fra. Dette sørger for at et tankobjekt blir tvunget til å implementere de metodene som kreves av resten av spillet, nemlig å kunne sette nivå og endre størrelse. Når størrelsen endres må nivået også endres. For å berengne det nye nivået må tanker kunne gi ut volumet av vannet i tanken og overflatearealet.



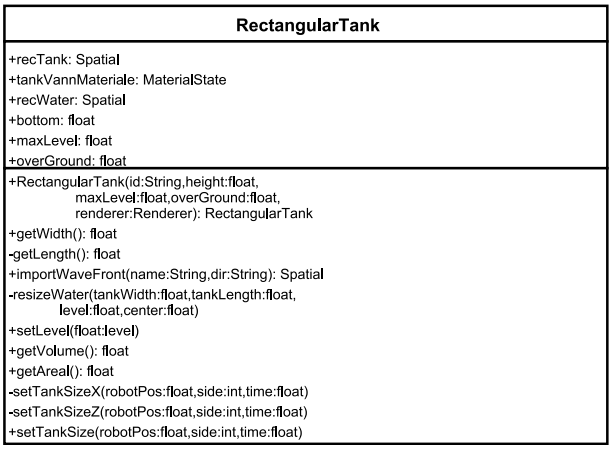
Figur 3.5: Klassediagram for klassen TankSystem



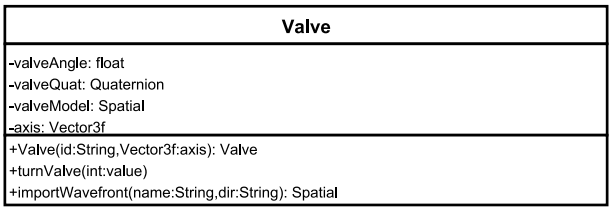
Figur 3.6: Klassediagram for den abstrakte klassen Tank

Klassen som realiserer den rektangulære tanken (fig. 3.7) som skal benyttes i spillet blir altså utvidet fra den abstrakte Tank-klassen. I en rektangulær tank kan vannet modelleres som en rektangulær boks. Nivået i vanntanken gis av simulatoren. Vannstrålen som kommer inn i tanken må skaleres riktig slik at den går fra rørutgangen til vannoverflaten.
For å kunne stille på mengden vann som går inn eller ut av tanken må det være mulig å stille på ventilrattet. Rollefiguren må kunne bevege seg bort til disse ventilrattene og manipulere dem. Når disse blir stilt på, må både mengden vann visuellt endre på seg, og reguleringsmodellen må få beskjed om endringen.

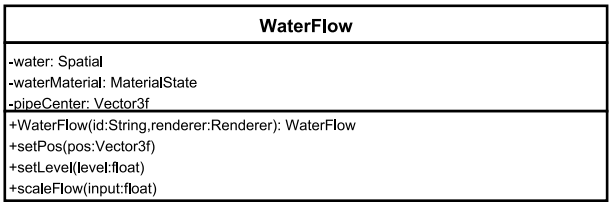
Siden det skal kobles på to ventilratt og to vannstråler (inn og ut) lages disse som objekter. Klassediagrammer for disse klassene vises henholdsvis i figur 3.8 og 3.9.



Figur 3.7: Klassediagram for rektangulær tank



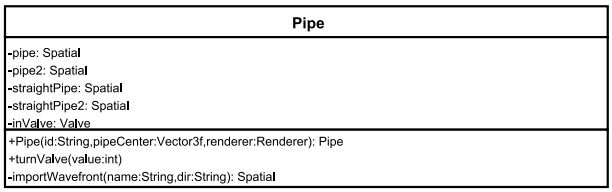
Figur 3.8: Klassediagram for ventilklassen



Figur 3.9: Klassediagram for vannstråleklassen

For å kunne endre størrelse på selve tanken, kan rollefiguren bevege seg opp til tanken, ta tak i den, og trekke i tanken for å gjøre den større. For å vite når rollefiguren er nær nok tanken for å ta tak, legges det ikke-taktile gjennomsiktige plater i en viss avstand fra tanken. Hvis rollefiguren er i kontakt med disse, er den tilstrekkelig nær tanken. Deretter kan mellomromstasten benyttes for å la rollefiguren gripe tak i tanken. Tanken vil bare bli forstørret i en retning.
For å ikke binde seg i hvordan vannrøret skal plasseres, deles dette opp i rette deler og bøyde deler. Dermed kan det settes sammen som man selv ønsker. De sammensatte

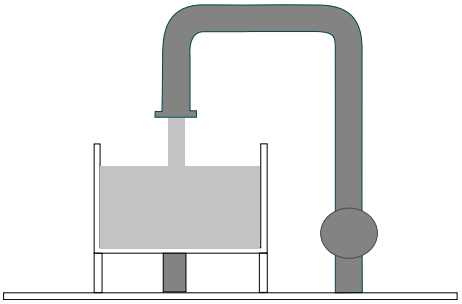
rørdelene plasseres i et objekt. Rattet til inngangsventilen plasseres også i dette objektet. Klassediagramet for rørsystemet er vist i figur 3.10.



Figur 3.10: Klassediagram for røret

3.5.5.1 Konsepttegning av tank

Tanken og systemet rundt denne er tenkt å se ut som i figur3.11.Selve tanken lages i blender 3D, mens vannet lages direkte i spillmotoren for å ha bedre kontroll på å kunne sette nivå og mengde inngangsvann.For å ikke binde seg i hvordan røret skal plasseres, deles dette opp i rette deler og bøyde deler. Dermed kan det settes sammen som man selv ønsker. Ventiler lages som ratt festet på røret. Også dette lages i Blender.

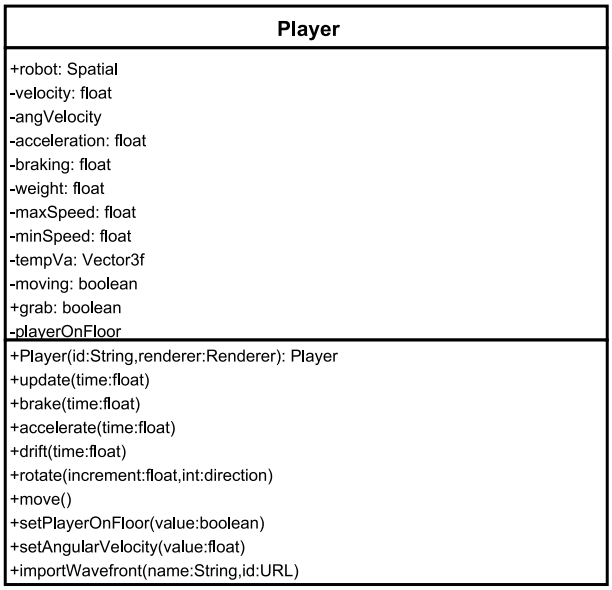


Figur 3.11: Ide til hvordan tanken skal se ut

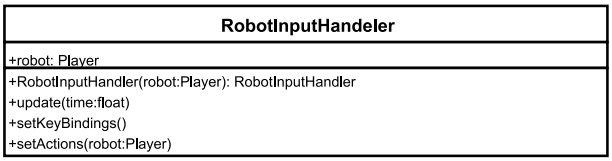
3.5.6 Rollefigur

Dette spillet skal hjelpe til med å gi en illustrasjon av et fysisk prinsipp. Det er derfor viktig at man får god oversikt over scenen, så man ser hva som foregår. For å kunne se scenen fra forskjellige vinkler velges det å la en rollefigur kunne bevege seg rundt i scenen. Den mest aktuelle måten å vise bildet på da er med en tredjepersons synsvinkel og følgekamera. For å realisere dette lages en håndterer for tastetrykk etter inspirasjon fra leksjon 9 som følger med i spillmotoren. Disse lar rollefiguren styres med tastene WSDA og musen, og benytter et følgekamera for vise scenen. Rollefiguren skal bevege seg slik at den aksellereres med tasten "'W"', rygger med tasten "'B"', og svinger til venstre og høyre med henholdsvis tastene "'A"' og "'D"'. Når tastene slippes, skal rollefiguren på kort tid senke farten ned til null. Når rollefiguren svinger skal fartsvektoren den ha endre retning til den nye retningen, dvs rollefiguren skal ikke skrense eller skli. Figur 3.12

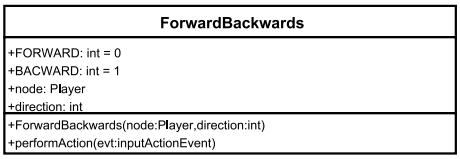
viser klassediagrammet for rollefiguren. For å kontrollere bevegelsene skal det brukes en håndterer, se fig. 3.13. Denne skal definere hendelsene forover, bakover og svinging, se fig. 3.14 og 3.15.



Figur 3.12: Klassediagram for rollefiguren



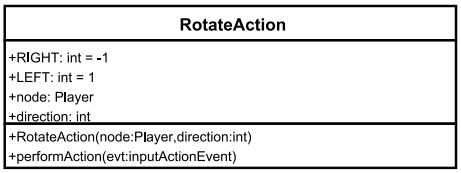
Figur 3.13: Klassediagram for tastetrykkhåndtereren



Figur 3.14: Klassediagram for forover- og bakoverhandlingen

3.5.6.1 Kollisjonssystem

For at rollefiguren skal ha en naturlig måte å bevege seg på, er det viktig at den ikke kjører gjennom andre objekter i scenen, men istedet kolliderer på en naturlig måte. Det

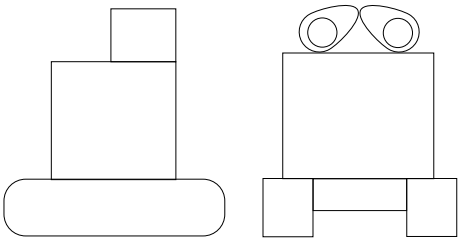


Figur 3.15: Klassediagram for roteringshåndtereren

viser seg at det innebygde kollisjonssytemet i jMonkey har noen feil, og ikke alltid registrerer kollisjoner. Av denne grunn blir jMonkeys fysikkmotor jMonkey Physics Engine tatt i bruk. Dermed kan rollefiguren implementeres som en dynamisk node, mens alle objekter som det skal kollideres i implementers som statiske noder. Dermed blir kollisjonene håndtert av fysikkmotoren.

3.5.6.2 Konsepttegning av rollefigur

Rollefiguren skal bevege seg rundt i et slags industriområde. For å la kompleksiteten og mengden animasjon av denne rollefiguren velges det å la denne være en enkel robot med beltehjul. En konseptidè til roboten er vist i figur 3.11. Robotens beltehjul kan animeres så det ser ut som om de går rundt. Eventuelt kan det lages en tekstur av beltehjul som er animert, for å gi en illusjon av bevegelse.

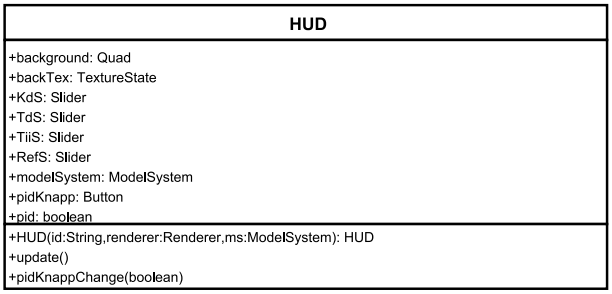


Figur 3.16: Ide til hvordan rollefiguren skal se ut

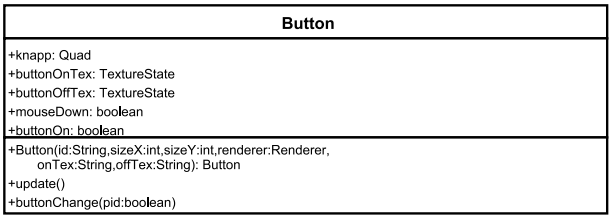
3.5.7 Styring av parametre

Nivået i tanken styres av en regulator. Det er ønskelig å la brukeren styre parametrene for denne regulatoren. For å la brukeren gjøre dette er det sett på flere metoder. Den første er å bruke java sitt swing-system som tilbyr en rekke bruktergrensesnitt som knapper, glidekontakter og så videre. Disse kan plasseres på et panel utenfor selve spillet. Ulempen med denne metoden er at klassene som sørger for visning av spillbildet tar utgangspunkt i størrelsen av programvinduet. Dersom størrelsen på programvinduet endres i forhold til spillbildet, fører dette til feil synsvinkel. En annen mulighet er å bruke det innebygde JMEdesktop, der man kan bygge opp et panel som så blir tegnet opp som en tekstur. Uheldigvis fungerer ikke dette med appletsystemet som blir benyttet, da ingen av brukergrensesnittkomponentene reagerer på museklikk.

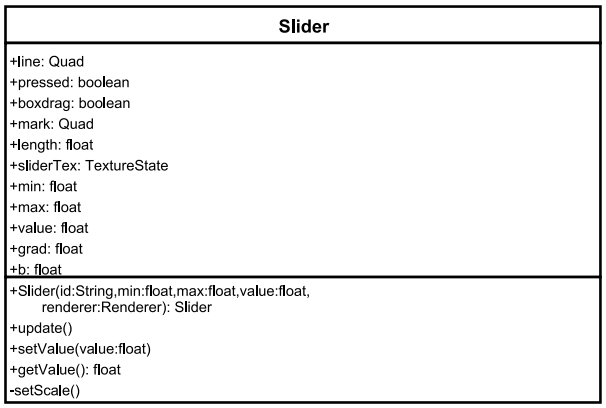
Den måten som realiseres i spillet er å konstruere en enkel HUD (se avsnitt 2.1.3.17). Denne kan inneholde knapper og glidekontakter for å kontrollere regulatorparametre, og skru av og på regulatoren. Ulempen med denne metoden er at knapper og glidekontakter må lages fra bunnen av. Klassediagramet for HUDen er vist i figur 3.17, og klassediagrammene for henholdsvis knapp og glidekontakt er vist i 3.18 og 3.19. Som nevt i avsnitt 3.5.5 kan rollefiguren endre størrelse på tanken og og inngangsstrømning til tanken. Når pid-regulatoren er skrudd av, skal det også være mulig å stille på ventilrattet som styrer utgangsstrømningen fra tanken.



Figur 3.17: Klassediagram for HUD



Figur 3.18: Klassediagram for knapper til HUD



Figur 3.19: Klassediagram for glidekontakter til HUD

3.5.7.1 Konsepttegning av HUD

HUDen vil bli realisert ved å la en plate bli rendret ortogonalt nederst på skjermen, og legge en tekstur på denne platen. HUDen skal inneholde glidekontakter for å endre regulatorparametrene og en knapp for å skru av og på regulatoren. I tilleg skal den kunne ha knapper for å velge hvilken form tanken skal ha. En ide om hvordan HUDen skal se ut vises i figur 3.20.

Figur 3.20: Ide til hvordan HUDen skal se ut

3.5.8 Alternativ kameraføring

En annen aktuell kameraoppførsel kan være å la kameraet være uavhenging av rollefiguren. Kameraet kan da enten stå i ro ett sted, eller man kan gjøre det mulig å også kontrollere kameraet. En mulighet er da også å gi brukeren muligheten til å veksle mellom disse to kameravinklene.
Kamera to kan settes til å kunne bevege seg i en sirkel rundt den aktuelle scenen, og styres med piltastene på tastaturet. For å kunne bytte mellom de to kameraene kan det være en fordel å sette opp to kameraer. Man kan så sette opp F1 og F2 tastene på tastaturet til å veksle mellom disse to kameraene.

Kapittel 4

Implementering av GUI for undervisningsspill

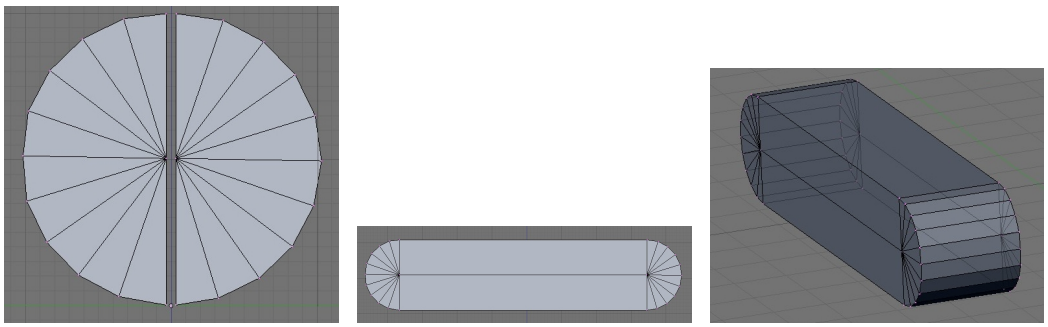
4.1 Grafikk

For å kunne se hvordan spillet ville oppføre seg ble 3D-modellene av gjenstandene som skal vises i testprogrammet utviklet tidlig. Modellene gikk gjennom en utvikling fra konsepttegningene.

4.1.1 Robot

4.1.1.1 Beltehjul

Beltehjulene lages ved å splitte en sirkel i to (se figur 4.1a, og deretter koble disse sammen med ekstra overflater som i figur 4.1b. Dermed har man formen på beltehjulet i 2D. Deretter ekstruderes denne formen, dvs. at figuren strekkes ut slik at den blir 3-dimensjonal, se figur 4.1c.

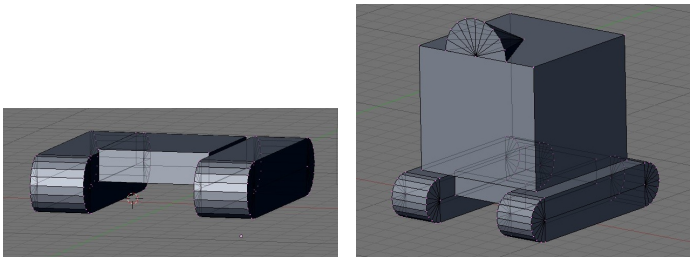


(a) Halvsirkler (b) Forlenget (c) Beltehjul i 3D

Figur 4.1: Tegning av beltehjul

Resten av figuren ble laget ved å duplisere beltehjulet og deretter sette dem sammen

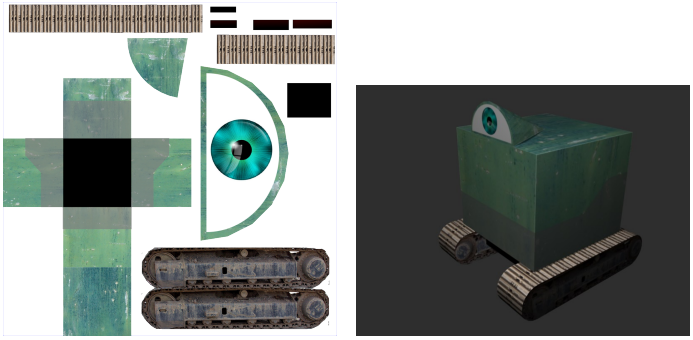
med en kube, se figur 4.2a. Nok en kube blir brukt som robotkroppen, mens en kjegle delt i to blir brukt som et øye, se figur 4.2b.



(a) understell (b) Ferdig modellert

Figur 4.2: Sammensetting av robot.

4.1.1.2 UV-kart og tekstur til roboten



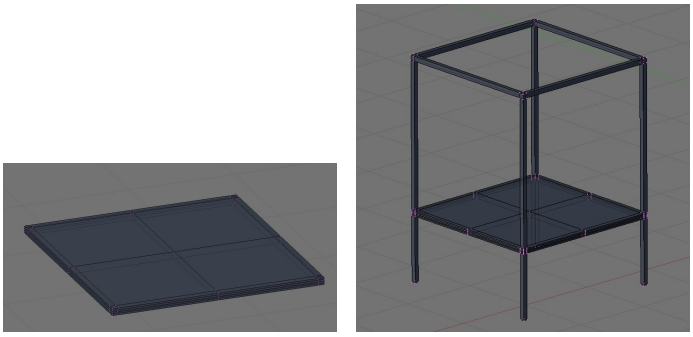
(a) Teksturfil (b) Bilde av roboten rendret i Blender

Figur 4.3: Tekstur på roboten

4.1.2 Tanksystem

4.1.2.1 Tank

For å få god innsyn til tanken var det opprinnelig tenkt å lage denne som et slags akvarium, altså en gjennomsiktig boks. På grunn av problemer med gjennomsiktighet (se avsnitt 4.1.2.2) blir tanken i stedet tegnet som en tank med kanter, se fig. 4.4b. Tanken ble laget ved å dele hver flate på boksen i 16 deler, og la hjørnene på flaten bli hjørnekanter og bein, se fig. 4.4a. Hjørnekantene ekstruderes så på toppen, og forlenges slik at de danner toppkantene. Når denne kanten møter en hjørnekant, kan man slå sammen to og to punkter slik at hjørnekant og toppkant er festet sammen

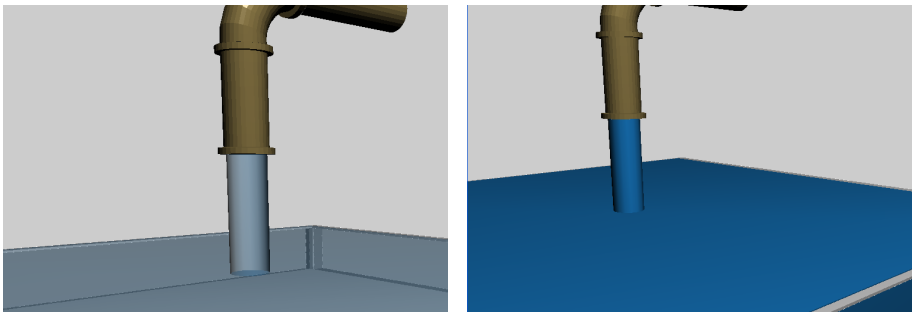


(a) Bunnplaten i karet (b) Karet med ferdige kanter

Figur 4.4: Rektangulær tank

4.1.2.2 Vannet

Vannet i tanken skal tegnes som en boks, og vann som strømmer inn og ut av tanken tegnes som sylindere. Det var planlagt å la vannet være gjennomsiktig, men det viste seg å være et problem med sylinderne. Det virker som om gjennomsiktige sylindere bare kan se gjennom ett "'lag"' av sylinderen, se figur 4.5a. Da de nye klassene for applets ble tatt i bruk, dukket det oppet nytt problem: Figurene som var gjennomsiktige ble usynlige. Det ble derfor valgt å la vannet være ugjennomsiktig.



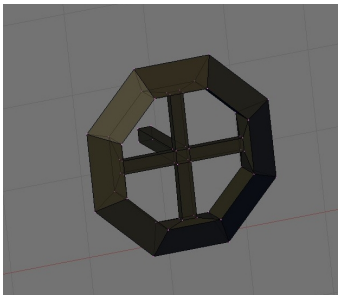
(a) Gjennomsiktighet skrudd på (b) Gjennomsiktighet skrudd av

Figur 4.5: Problem med gjennomsiktighet

4.1.2.3 Rør

4.1.2.4 Ventil

Ventilen modelleres i Blender som et ventilratt som kan skrues på. Ventilrattet er vist i figur 4.6



Figur 4.6: Ventilratt

4.1.3 Gulv og omgivelser

Gulvet som rollefiguren skal kjøre på lages som en boks direkte i jMonkey. Den får lagt på seg en tekstur av et industrigulv. Gulvet kobles til en statisk node i fysikkrommet, og rollefiguren kan dermed bevege seg oppe på det. I tillegg lages det en boks som kobles til en dynamisk node. Denne legges slik at rollefiguren kan kjøre på den, og vise at fysikkmotoren fungerer.

4.1.4 HUD

HUDen lages som et område av skjermen der brukeren kan få informasjon om systemet, samt endre parametre for systemet. For dette spillet er det lagt inn en knapp for å kunne skru av og på pid-regulatoren, og glidekontakter for å endre parametrene i denne regulatoren. HUDen skal tegnes opp som vist i figur 4.7.

Figur 4.7: Ferdig HUD

Dette panelet lages ved å legge en tekstur på et rektangulært plan. På panelet kan knapper og glidekontakter plasseres. Knapper er også laget ved med et rektangulært plan. Knapp-objektet har en oppdateringsfunksjon som undersøker om musen har blitt klikket på den ved å se om musepekeren er innenfor planets grenser når det klikkes. Hvis den blir klikket på, byttes det mellom to teksturer som viser henholdsvis en avslått og en påslått knapp. Glidekontakten består av to rektangulære plater. Den ene viser banen som kontakten kan bevege seg i, mens den andre er selve kontakten som kan flyttes på. Objektets oppdateringsfunksjon skjekker om musen har blitt klikket innenfor denne kontakten, og hvis musen holdes inne kan man dra glidekontakten til ønsket posisjon. Ved oppsett av glidekontakten settes mininumsverdi og maksimumsverdi. Disse brukes til å lage en lineær funksjon. Glidekontakten har en satt lengde på 100. Stigningsgraden (grad) og krysningspunktet med y-linjen (b)i den lineære funksjonen der 0 på glidekontakten tilsvarer minimumsverdien og 100 er maximumsverdien kan finnes med funksjonen

setScale().

```
private void setScale(){
grad = length/(max-min);
b = length/2 -(grad*max);
}
```

HUDden skal rendres uten perspektiv, dvs ortogonalt. Dette gjøres ved å be renderen om å ikke tegne opp denne noden med funksjonen setRenderQueueMode(Renderer.QUEUE_SKIP). I hovedfilen benytter man funksjonen simpleRenderer(), som lar programmereren legge inn oppgaver til renderen. Her endrer vi rendretypen til ortogonal, ber om å tegne opp HUDen og setter renderen tilbake til perspektiv-tegning. I tillegg settes en parameter for denne noden som gjør at rendreren tegner den opp med jevnt lys, dvs at lyset påvirker den likt uavhengig av hvilken vinkel lyset treffer den med.
Verdien som glidekontaktene har i øyeblikket var tenkt å være skrevet ved siden av glidekontakten ved hjelp av tekst-klassen til jMonkey. Det viser seg imidlertid at det nye appletsystemet ikke får vist tekst, og denne blir derfor ikke lagt på.

4.2 Oppsett av spill

For å lage et spill av typen SimplePhysicApplet må metoden SimpleInitGame eksistere. I denne metoden kan alle objekter som skal være i spillet initialiseres. Initialiseringen er som implisert i figur 3.2 delt inn i flere metoder for å gi bedre oversikt. Av disse er det setupPhysics, setupChaseCamera og setupLight som legger grunnlaget for scenen. setUpPhysics genererer et fysikkrom. I tillegg legger den til funksjonalitet i dette fysikkrommet som er nødvendig for rollefigurens styreingssystem (se avsnitt4.4). Metoden setupChaseCamera initaliserer et følgekamera som skal følge etter rollefiguren. Dette kameraet kan settes opp med instillinger for hvordan det skal bevege seg i forhold til rollefiguren. Instillingene går på hvor langt unna og hvor nærme kameraet kan være, innenfor hvilke vinkler det skal se på rollefiguren osv. I setupLight settes det opp et punktlys.

4.3 Spilløkken

Når spillet er satt opp går programmet inn i spilløkken. Her blir tastetrykk og lignende som er registrert i tastetrykkhåndtereren tatt i mot og behandlet. I programmet har funksjonen simpleUpdate() muligheten til å legge inn oppdateringer for de forskjellige objektene i spillet.

4.4 Styring av Rollefigur

Styringssystemet til rollefiguren er basert på leksjon 9 som følger med i kildekoden til jMonkey engine. Rollefiguren styres av en klasse som kalles RobotInputHandler. Denne klassen sørger for at tastene WSAD lar brukeren styre rollefiguren. "‘W"’ kobles mot metoden "‘accelerate"’ i rollefigurens klasse, "‘S"’ kobles mot "‘break"’ mens A og D

kobles mot metoden "'rotate"'. Alle disse funksjonene tar inn tiden som har gått siden siste gjennomkjøring av spilløkken. Robotklassen har en satt akserlerasjon for foroverbevegelse og en satt akselerasjon for rygging. Farten til roboten blir satt av formelen for lineær akselelerasjon, begrenset av en maksimal hastighet.

```
public void accelerate(float time) {

velocity += time * acceleration;
if (velocity > maxSpeed) {
velocity = maxSpeed;
}

}
```

Tilsvarende reduserer funksjonen "'break"' hastigheten ned til en minimumshastighet, som er negativ og derfor lar rollefiguren rygge.

For å simulere friksjon benyttes funksjonen "'drift"'. Denne funksjonen fungerer på samme måte som akselereringen og bremsingen, men reduserer hastigheten ned mot null så lenge absolutthastigheten er over en terskelverdi. Denne funksjonen blir kalt av oppdateringsfunksjonen til RobotInputHandleren som igjen blir kalt for hver gjennomkjøring av spilløkken.
Ved å trykke kjøre "'rotate"' funksjonen tas retningen som det ønskes å rotere inn som -1 eller 1. Rollefiguren har en satt rotasjonshastighet som blir ganget med denne hastigheten. For hver gjennomkjøring av spilløkken blir så oppdateringsfunksjonen for rollefiguren kjørt. Før denne gjennomkjøringen av spilløkken har fysikkmotoren fått lagt inn muligheter for å gjøre endringer i parametre før fysikkberegninene blir gjort. Her settes vinkelhastigheten på rollefiguren til null, i tillegg til at en parameter som forteller om rollefiguren er i kontakt med gulvet blir satt til usann. Når fysikken blir beregnet, er det lagt inn en test som skjekker om rollefiguren faktisk er i kontakt med bakken ,og setter denne til sann hvis dette er tilfellet. På grunn av litt vanskeligheter med den fysiske modelleringen av bevegelsen til rollefiguren (se avsnitt 5.1.4) er det også lagt inn en verdi som er sann så lenge roboten er i bevegelse. Hvis den ikke er i bevegelse, eller roboten ikke er i kontakt med gulvet, bli rollefiguren påvirket av gravitasjonen som er innebygget i fysikkmotoren.

```
public void update(float time) {
DynamicPhysicsNode temp = (DynamicPhysicsNode) this.getParent();

if (playerOnFloor&&moving) {
temp.setAngularVelocity(angVelocity);
Vector3f velocityVect = temp.getLocalRotation().getRotationColumn(
2, tempVa).multLocal(velocity);
temp.setLinearVelocity(velocityVect);
}

}
```

Som metoden viser, setter man rotasjonshastigheten på den dynamiske noden til rotasjonshastigheten fra "'rotate"'-funsjonen. Deretter finner man en vektor som peker i

den retningen roboten peker i, og ganger denne med hastigheten man har fått gjennom akselererings- eller bremsefunksjonen.

4.4.1 Kollisjonshåndtering

Som nevnt i designkapittelet blir kollisjoner håndtert av fysikkmotoren. For å benytte seg av dette blir alle gjenstander som skal være statiske lagt under en statisk node, mens alle noder som skal bevege seg legges under dynamiske noder. Objektene vil etter dette kollidere på en naturlig måte.

4.5 Styring av tanksystemet

Styringen av tanksystemet begynner med at det undersøkes om ventilene skrus mer opp eller igjen. En markør er plassert ved området der ventilrattene er. Ved å benytte den innebygde kollisjonstesten i jMonkey kan man finne ut om rollefiguren er innenfor dette området. En funksjon tester dette, og skjekker samtidig om brukeren har trykket på venstre eller høyre piltast. Dersom dette er tilfelle, blir den nye verdien på åpningen lagret. Denne blir så sent til modellsystemet, og brukes samtidig til å rotere ventilrattet tilsvarende mye. Dersom pid-regulatoren er skrudd av tillates det at man skrur på utgangsventilen, som skjekkes på samme måte. Neste skritt er å hente ut verdier fra modellsystemet; inngangsstrømning, utgangsstrømning og nivå i tanken. Disse tre verdiene blir matet inn i tankstyringsystemet. Inngangs- og utgangsstrømningen endres visuellt ved å skalere i x- og y-retning.
For å sette nivået i den rektangulære tanken kan boksen som representerer vannet skaleres i høyderetning. Når et objekt skaleres i jMonkey blir det strukket i både positiv og negativ retning. Ønskes det å bare skalere i en retning må man dermed både skalere og flytte objektet. Dette blir gjort ved hjelp av funksjonen setLevel.

```
public float setLevel(float level){
 if(level<=0) level = 0;

 Vector3f scale = recWater.getLocalScale();

 scale.y = level;
 recWater.setLocalScale(scale);
 recWater.updateModelBound();

  BoundingBox bb = (BoundingBox)recWater.getWorldBound();
 float height = bb.yExtent;
 Vector3f pos = recWater.getLocalTranslation();
 recWater.setLocalTranslation(pos.x, bottom+height, pos.z);

 float vannSpeil = bottom+(2*height);
    return vannSpeil;
}
```

Vannet i tanken er modellert av en boks, og nivået settes ved å skalere denne boksen i y-retning. Hvis bare dette ble gjort, ville vannboksen komme ut på undersiden av tanken,

og nivået hadde heller ikke blitt riktig. Derfor må den også flyttes opp slik at bunnen av vannboksen er på bunnen av tanken. Posisjonen til vannoverflaten sendes så videre til skaleringsfunksjonen til inngangsstrømningen. Denne sørger for at vannstrålen på samme måte blir skalert i høyden slik at den går fra utgangen av røret til vannoverflaten.

4.5.1 Skalering av tank

For å skalere tanken, testes det først om rollefiguren er nærme nok en kant til å trekke i tanken. Til dette brukes en gjennomsiktige, ikke-taktile plater som legges i en avstand fra tanken . Kollisjonssystemet i jMonkey benyttes for å teste om rollefiguren er nær nok tanken til å gripe tak i den. Hvis brukeren trykker ned mellomromstasten, griper rollefiguren tak i tanken.

Skaleringen av tanken foregår så ved å regne ut avstanden mellom kanten av rollefiguren og kanten av tanken, og deretter flytte tanken halvparten av denne avstanden mot rollefiguren. Deretter finnes faktoren tanken skal skaleres med ved å dele tankens gamle lengde på den nye. Denne faktoren må så ganges med tankens nåværende skaleringsfaktor for å finne den endelige nye faktoren. Dette kan sees ved et eksempel. Man har en boks på 1 ganger 1 meter, som så skaleres med 4. Hvis man ønsker å skalere denne nye boksen med 2 igjen, må man sette skaleringsfaktoren til 8, som nettopp er den gamle skaleringsfaktoren ganget med den nye.
Etter at skaleringen av tanken er gjennomført blir platene som brukes som markører for at rollefiguren er nær tanken flyttet etter slik at avstanden til tanken blir opprettholdt.
For å sørge for at nivået i tanken blir endret når tankstørrelsen endres, må modellsystemet oppdateres med den nye tankstørrelsen. Denne er lagret i modellsystemet som arealet av vannoverflaten. Ved å dele det nye arealet av den grafiske vannflaten på det gamle arealet får vi faktoren som tankarealet i modellsystemet må ganges med for å få det nye arealet. Modellsystemet blir oppdatert med denne nye verdien.

Skalering av statisk fysikknode

Et problem dukket opp da tanken skulle skaleres. Statiske fysikknoder er ikke beregnet for skalering. Når man endrer størrelse på disse må man fortelle fysikkmotoren at det har skjedd endringer med funksjonen `generateGeometry()`. Det denne gjør er å legge til den geometrien som er koblet til fysikknoden til det som fysikken skal beregnes fra. Dermed blir både den opprinnelige og den nye geometrien liggende i fysikknoden. For å løse dette skrus fysikkberegningene for tanken av mens den skaleres. I det rollefiguren slipper tanken, slettes all geometri under fysikknoden, for deretter å legges til igjen. Når `generateGeometry()` nå kjøres, er det kun den oppdaterte geometrien som benyttes i fysikkberegningene.

4.6 Effekter - Skygge

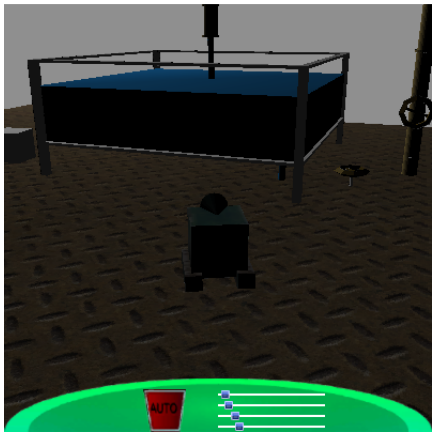
Skygge settes opp i jMonkey ved å lage en ny render passering. I testsystemet lages denne i SimplePhysicsApplet-klassen. Deretter kan man fortelle denne passeringen hva som skal kaste skygge. Generering av skygge krever en del av prossessoren, så det er kun valgt å la roboten kaste skygge. I tillegg må lyskilden som benyttes settes som skyggekastende lyskilde.

4.7 Start av applet

For å kunne kjøre programmet i en nettleser må det pakkes i jarfiler som signeres som i A.5. Deretter må det lages en nettside der disse pakkene lastes inn dette ble gjort som beskrevet i vedlegg A.4. Her benyttes et metode fra LWJGL for å laste inn alle biblioteksfiler som trengs til programmet før selve appleten starter. Biblioteksfilen som behøves for å benytte fysikkmotoren blir ikke lastet inn her. Det ble gjort forsøk på å finne ut hvordan denne skal kunne lastes inn uten hell. For å få mulighet til å kjøre programmet, må denne biblioteksfilen legges et sted hvor appleten får tilgang til den. I windows kan dette for eksempel være mappen c:/windows/System32.

Kapittel 5

Evaluering av det utviklede systemet



Figur 5.1: Kjøring av testsystemet

Det utviklede testsystemet ligger på cden som er lagt ved denne oppgaven. På grunn av problemer med innlasting av fysikkmotoren må biblioteksfilen for denne legges manuelt et sted appleten kan få tak i den. (Se avsnitt 4.7). For å kjøre appleten etter dette åpnes htmlfilen i vedlegg A.1.

5.1 Test av systemet

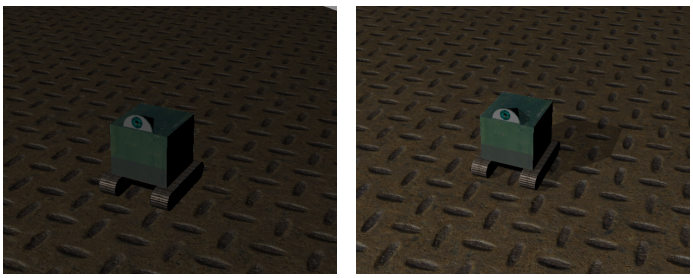
5.1.1 Enkelhet - hvor lett er det å bruke motoren

Under uttesting av motoren ble det testet å lage vanlige spill (ikke applets). Dette fungerer meget intuitivt, ved å opprette objekter med initialiserngsmetoden og oppdatere dem i oppdateringsfunksjonen. Når det gjelder applets er det en del problemer når det gjelder lasting av bibliotekfiler. Klassene laget av brukerne på forumet løser dette delvis,

men det krever allikevel endring av motoren for å ta i bruk. Når dette er tatt i bruk er spillutviklingen imidlertid like enkel som for vanlige spill. Å legge til objekter fra eksterne redigeringsprogrammer gjøres med enkle fuksjoner. Klassene gjør det mulig å starte et spill med en tom scene uten noen å programmere noe kode. Problemet med kollisjonsdeteksjonen i motoren gjør at spillet må kompliseres noe med fysikkmotoren. Det som gjør dette til et problem er ønsket om å kunne skalerere statiske objekter, noe som det ikke er lagt til rette for i motoren. Løsningen med å ignorere fysikken fra tanken, fjerne fysikknoden fra systemet for å så legge den til igjen er ikke elegant, men løser problemet. Det bidrar imidlerid til et mer komplisert system. I tillegg lastes ikke biblioteksfilene til fysikkmotoren automatisk, noe som gjør hele systemet vanskelig å benytte for brukeren.

5.1.2 Grafikk

Scenen som tegnes opp i spillet ble gjennom designfasen endret en del. En av grunnene for dette var problemer med gjennomsiktige objekter. Vannet i tanken, som var tenkt å være gjennomsiktig, var fra enkelte vinkler ikke gjennomsiktig. Sylindere som skulle illustrere gjennomsiktige vannstrålene ikke gjennomsikige, se figur 4.5. Dette problemet ble enda verre med de nye appletklassene fra forumet(ref [10]), da gjennomsiktige objekter ikke ble tegnet i det hele tatt. Dette ble løst ved å tegne vannet ugjennomsiktig. Dette problemet med gjennomsiktighet er en svakhet ved jMonkey. I spillet er det også lagt til skygge på rollefiguren. Denne skyggen ble kun vist på Maskin 2 som kjører linux. Eneste forskjellen på programmene er her biblioteksfilene som lastes inn av programmet, så det er antatt at denne relativt store forskjellen kommer av dette. Se figur 5.2a og 5.2b.



(a) Maskin 1 (b) Maskin 2

Figur 5.2: Grafisk forskjell i Windows og Linux.

5.1.3 Bildefrekvens kontra oppløsning

Spillet ble testet på maskin 1 og maskin 2. Maskin 2 har mindre ressurser enn maskin 1. Spillene blir testet i økende oppløsninger, fra 200x200 og oppover. Maskin 2 klarer 300x300, men bildefrekvensen blir svært dårlig ved 400x400. Maskin 1 klarer fint å kjøre spillet i fullskjerm (1080x1024), og har dermed ikke problemer med dette. Maskin 2 får i tillegg problemer når oppløsningen går over 400x400, fordi fysikkmotoren er satt til en

minimum oppdateringsfrekvens på 5 , noe som den på denne oppløsningen har problemer med å holde. Tabell 5.1 gir en oversikt over gjennomsnitts bildefrekvens for forskjellige oppløsninger. Fordi maskin 1 ikke klarer å vise skygger, får den en fordel i denne testen, da dette er en krevende operasjon. For å se hvordan den klarer seg uten skygger, blir testen gjennomført en gang til med skygger avslått for maskin 2.

Tabell 5.1: Bildefrekvens med hensyn på oppløsning

	200x200	**300x300**	**400x400**	**1024x1080**
Maskin 1	100	100	100	60
Maskin 2	20	10	6	N/A
Maskin 2 uten skygge	30	17	10	

5.1.4 Kontrollering av spillet

Det utviklede styringssystemet for rollefiguren lar spilleren flytte rollefiguren rundt på hele spillebrettet. Det er allikevel gjort noen kompromisser for å få en god bevegelse men allikevel benytte seg av fysikkmotoren. Rollefiguren påvirkes av fysikken, men påvirkes bare av gravitasjonen når den står stille, eller når den har kjørt utenfor kanten av gulvet. Det vil si, at dersom den holder på å velte, kan den bli værende i en nesten veltet tilstand inntil spilleren slutter å bevege rollefiguren. Dette kan virke rart og unaturlig.
Kommunikasjonen mot spillmotoren har fungert bra, det har vært enkelt å få tak i verdier fra simulatoren, og sende verdier inn. Et problem her er at simulatoren til Cyberlabs må stoppes før man kan endre arealet i tanken. Dette er imidlertid en oppgave som antageligvis er lett å løse, og spillet har også vist at løsningen med å stoppe og starte simulatoren fungerer.
Appletsystemet kan ikke benytte seg av jMonkeys innebygde system for å benytte seg av Swing-komponenter som knapper og glidekontakter. Løsningen med å lage sine egne knapper fungerer, men det nye appletsystemet har også problemer med å vise tekst. Det er dermed vanskelig å vise informasjon om verdier som tekst på spillbildet.

Kapittel 6

Forslag til videre arbeid

Denne oppgaven har vist at det er mulig å å benytte jMonkey sammen med Cyberlabs simuleringsystemer. Det er imidertid rom for forbedringer av det utviklede systemet.
Den største ulempen med spillet slik det nå fungerer er at det er avhengig av fysikkoblingen som bruker fysikkmotoren ODE. Denne krever at biblioteksfilen blir lastet inn, noe som i denne oppgaven må gjøres manuelt. Ved å forsøke å bytte ut det c++ baserte ODE med javaekvivalenten JOODE, kan det bli mulig å lage spill av denne typen uten avhengigheten mot biblioteksfilene. I tillegg er styringen av rollefiguren i dette spillet ikke optimal. Ved å bygge opp et styringssystem der bevegelsene er bygget på å la rollefiguren bli påvirket av krefter vil bevegelsene bli mer naturlige. Hvis man ønsker å lage 3D-spill må man være klar over at dette er mer krevende for datamaskinene enn enkle 2D-spill. Hvis disse spillene skal brukes i undervisningssammenheng er det viktig at brukere kan kjøre spillet. Det kan derfor være lurt å undersøke om det er mulig å redusere operasjonene som må gjøres i spilløkken, og dermed øke bildefrekvensen.
jMonkey er stadig under utvikling. Løsningen for applets som jeg har tatt i bruk i denne oppgaven, og som er beskrevet i [10], er når denne oppgaven leveres blitt godkjent og dermed lagt inn permanent i motoren. Fysikkmotoren som er benyttet har gitt noen utfordringer i programmet. Hvis det innebygde kollisjonssystemet i motoren blir bedre, kan man unngå å benytte fysikkmotoren i det hele tatt. På denne måten kan man både få ned krav til prossessering, og unngå bruk av ODE. Det er uansett en god ide å benytte den nyeste utgaven av spillmotoren.
Dersom man ønsker å benytte ode, må det jobbes frem en løsning for å kunne automatisk laste inn biblioteksfilene som trengs for å starte appleten. Appleter har begrensninger når det gjelder minne og tilgang på datamaskinen. Hvis spillene som skal utvikles blir større kan det derfor også være aktuellt å lage dem som Java Webstart-programmer istedet, se avsnitt 2.3.2.

Kapittel 7

Konklusjon

Målet for denne oppgaven har vært å teste ut en spillmotor for å lage et 3D-GUI for Cyberlabs sine simulatorer. Det ferdige programmet skulle kunne kjøres som en java applet. For å få en forståelse for hvordan en spillmotor fungerer, er det gjennomført et studie av slike motorer. Ut i fra en rekke maskiner basert på åpen kildekode ble en rekke motorer valg ut som aktuelle. Gjennom å vurdere disse spillmotorene opp mot hverandre ble jMonkey Engine valgt ut som den mest egnede. Dette var særlig på grunn av at motoren har et stort antall brukere, som gjennom motorens forum kan bistå med hjelp til spørsmål rundt motoren. Motoren er også basert på java, og kan derfor lett brukes sammen med Cyberlabs simulatorer. Et testprogram ble designet og implementert, der man viser et tanksystem med inngangsstrømning og utgangsstrømning, og gir brukeren mulighet til å endre parametre for systemet. Det ferdige testprogrammet lar brukeren flytte en rollefigur rundt i en scene, og gir en visuell tolkning av simulatoren. Oppgaven viser dermed at jMonkey kan brukes til dette formålet. Det er allikevel avdekket problemer ved bruk av biblioteksfiler, og programmet har fremdeles forbedringsmuligheter. Dersom man ønsker å benytte et slik system for å lage en 3D-GUI, må man ta i betraktning at det er mer ressurskrevende for datamaskiner, og derfor vurdere om dette er et akseptabelt at mindre kraftige datamaskiner ikke skal kunne kjøre pogrammet.

Bibliografi

[1] ASG. Adventure game studio. http://www.adventuregamestudio.co.uk/, Juni 2009.

[2] Erwin Coumans. Bullet 2.74 physics sdk manual, 2009. [Online; accessed 13-June-2009].

[3] Keith Ditchburn. Vertex shader. http://www.toymaker.info/Games/html/vertex_shaders.html, 2009.

[4] Helge Foerster. jpct, June 2009.

[5] Garret Foster. Understanding and implementing scenegraphs. http://www.gamedev.net/reference/programming/features/scenegraph/, Mars 2009.

[6] Marvin Fröhlich. Xith3d, June 2009.

[7] Helmut Garstenauer. A unified framework for rigid body dynamics. Master's thesis, Johannes Kepler Universitët Linz, 2006.

[8] Nikolaus Gebhardt. Irrlicht, June 2009.

[9] jezek2. Jbullet - java port of bullet physics library. http://jbullet.advel.cz/, Juni 2009.

[10] jMonkey. Bullet 2.74 physics sdk manual, 2009. [Online; accessed 13-June-2009].

[11] Clas Mehus. Vertex shader. http://www.idg.no/pcworld/tips_og_guider/maskinvare/annet/article30837.ece, September 2008.

[12] Sun Microsystems. Java 3d parent project, 2009.

[13] Larry O'Brien. Introduction to directx's direct3d 10. http://developer.amd.com/documentation/articles/Pages/7112007172.aspx, 2007.

[14] Collins og Anshuman. Soft body dynamics. http://vizproto.prism.asu.edu/classes/sp03/wyman_g/SoftBodyDynamics.htm, Mars 2009.

[15] Calvin Austin og Monica Pawlan. *Advanced Programming for the Java 2 Platform.* Addison-Wesley, 1999.

[16] Stefan Zerbst og Oliver Duevel. *3D Game Engine Programming*. Thompson Course Technology, 2004.

[17] Eric Haines og Tomas Akenine-Möller. *Real-Time Rendering*. A.K. Peters Ltd., 2 edition, 2002.

[18] Mark Powell. jmonkey engine, June 2009.

[19] Alessandro Rigazzi. Optimizations with geometry shaders. `http://www.slideshare.net/acbess/geometry-shader-presentation`, Mai 2008.

[20] Russell Smith. Open dynamics engine. `http://www.ode.org/ode-latest-userguide.html`, Februar 2006.

[21] Sourceforge. Joode. `http://joode.sourceforge.net/`, April 2007.

[22] Jeff Ward. What is a game engine? `http://www.gamecareerguide.com/features/529/what_is_a_game_.php`, April 2008.

[23] Wikipedia. Java 3d api tutorial, 2000.

[24] Wikipedia. Blender (disambiguation) — wikipedia, the free encyclopedia, 2008. [Online; accessed 23-August-2008].

[25] Wikipedia. Adobe flash — wikipedia, the free encyclopedia, 2009. [Online; accessed 27-July-2009].

[26] Wikipedia. Bsd licenses — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=BSD_licenses&oldid=295204414`, 2009. [Online; accessed 8-June-2009].

[27] Wikipedia. Game engine — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Game_engine&oldid=297386734`, 2009. [Online; accessed 19-June-2009].

[28] Wikipedia. Gnu general public license — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=GNU_General_Public_License&oldid=297795591`, 2009. [Online; accessed 21-June-2009].

[29] Wikipedia. Gnu lesser general public license — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=GNU_Lesser_General_Public_License&oldid=293486570`, 2009. [Online; accessed 31-May-2009].

[30] Wikipedia. Hud (video gaming) — wikipedia, the free encyclopedia, 2009. [Online; accessed 5-July-2009].

[31] Wikipedia. Java web start — wikipedia, the free encyclopedia, 2009. [Online; accessed 3-June-2009].

[32] Wikipedia. List of game engines — wikipedia, the free encyclopedia, 2009. [Online; accessed 30-July-2009].

[33] Wikipedia. Mit license — wikipedia, the free encyclopedia, 2009. [Online; accessed 14-July-2009].

[34] Wikipedia. Shader — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Shader&oldid=297534755`, 2009. [Online; accessed 20-June-2009].

[35] Wikipedia. Zlib license — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Zlib_License&oldid=293995375`, 2009. [Online; accessed 2-June-2009].

Tillegg A

Innhold av CD

A.1 Testsystem: Applet

Ligger i mappen "'flow"'. Startes ved å kjøre appletloader.html i en nettleser. Filen odejava.dll som ligger i denne mappen må legges i c:/windows/system32.

A.2 Testsystem: Applet - kildekode

Ligger i mappen "'kildekode"'

A.3 UV-mapping i Blender

Filen: UV-tutorial.pdf

A.4 Installering av jMonkey Engine og jMonkey Physics

Filen: installjMonkey.pdf

A.5 Forbereding av Jar-filer

Filen:Applet.pdf