

# Funksjonalitett:

* Fornuftig Engine-struktur med inspirasjon fra forelesningene.
* Input fra tastatur og Mus
* Konfigurerbart spillvindu, både fra XML og funksjonskall på runtime
* Frame rate independant movement
* WASD for bruker-input. Alle spilltaster er konfigurerbare i XML
* Frame rate independent Animert 2D-grafikk
* Klient/Server-oppsett for Server og Klient via TCP-Protokollen hvor både server og klient kan styre hver sin figur på skjermen.
* Savegames med XML.

# Ekstra Funksjonalitet:

* Avansert grafikk-laster(Surfaces.h)
* Avansert XML-parser for lasting av settings, constants, savegames og andre resurser
* Videoavspilling vha. SMPG lib.
* Lydeffekter og Musikk vha. OpenAL lib.
* Meny Hvor man velger om man skal være Host eller Klient samt skrive inn IP(app\_name)
* Automatisk Garbage Collection(GCPtr)
* Logger med egene nivåer i XML for de individuelle klassene, lastes av default fra XML.
* Collision Detection for rects
* Timer med Sleep-funksjonalitet for å spare CPU
* Fysikk-klasse(Vector2D) med operator overlastninger til bruk som velocity, akselerasjon, gravitasjon og posisjon.
* Visning av FPS i application\_name
* Lagt opp for enkel spilloppbygging i GameManager
* Delvis optimalisert net-forbindelse ved sending pakker så skjeldent som nødvendig.

Litt urelatert:

* Profesjonell Engine-logo og Intro-video.

# Manglende funksjonalitet:

* Fungerende AI
* Støtte for flere enn 2 spillere (net klassene støtter det, men det er ikke implementert)
* Fullføring av spill
* Rom for en del optimalisering.
* Per pixel collision
* Exception om client ikke finner valgt host IP. Isteden henger man en stund før spillet starter som singleplayer.

# Utføring:

Gjennom utviklingen av dette prosjektet har vi Jobbet hver for oss med kommunikasjon via Skype, i tillegg til å jobbe en del hvor vi sitter sammen på 1 PC for de vanskeligere delene av prosjektet. Vi har fordelt arbeidsoppgavene ved hjelp av en Todo-list hvor vi har krysset av og lagt til emner ettersom de har blitt løst eller har kommet opp med nye ideer, og dette har resultert i bemerkelsesverdig lik arbeidsmengde fordelt oss imellom.

For å samkjøre prosjektene har vi brukt AnkhSVN satt opp mot <http://codesion.com/> sin SVN server.

Vi har også hatt god nytte av Visual Assist X for fortgang i kodingen.

# Struktur & Design:

Som nevnt, følger engine strukturen vår eksemplene fra forelesningen.

Main() oppretter først et kernel objekt, og legger deretter nødvendige Core tasks til i kernel loopen. Hver gang man legger til en task i kernelen, vil en Start() bli kalt på objektet. Så lenge det er tasks i kernel sin container med tasks, vil disse bli loopet igjennom og kalt Update() på. Om kernelen tømmes for tasks, avsluttes spillet.  
  
Tasks kan også legges til og fjernes på runtime. I Tillegg kan man pause og unpause en task. En task som er pauset vil bli liggende i kernel sin update loop, men Update() blir ikke kalt før tasken er unpauset igjen.

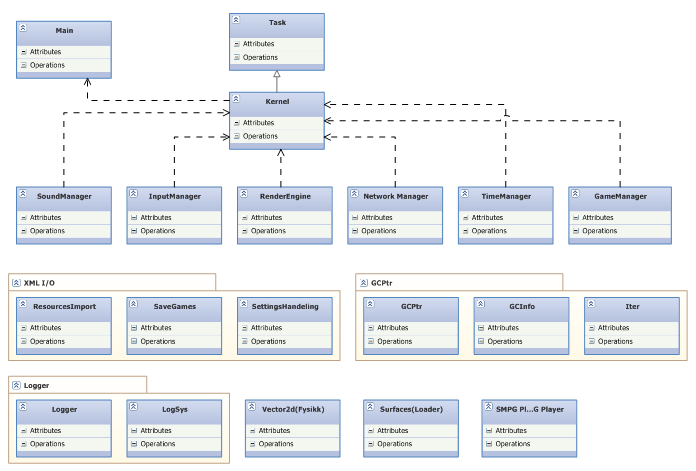
Klasser som det kun skal være ett objekt av bruker Singleton design pattern.

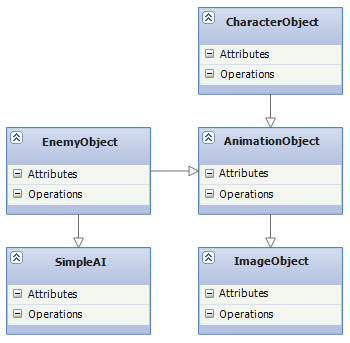
Vi bruker GCPtr<> (Garbage Collector) fra innlevering 1 i prosjektet. Dette både hindrer memory leaks, og korter drastisk ned på kodemengden i enkelte klasser, da man ikke trenger en milevis lang destructor med deletes.

Nettverk klassene våre er satt sammen av flere moduler hvor NetManager fungerer som et «wrapper». Dette gir gode forutsetninger for å utvide og gjenbruke nettverksbiten av enginen. Om man vil legge til nye typer messages som skal kunne sendes, trenger man kun å opprette en struct, legge til en verdi i enumen med message typer og legge den nye enum verdien til i to switcher.

Slik det er satt opp nå er det NetManageren som håndterer alle character objekter man mottar informasjon om fra en remote host/client. I tillegg trenger man kun gi NetManageren en referanse til en container med character objekter i, og den vil kunne ta seg av å sende oppdateringer om dine characters til hosten/clienten man er connected til. Om man heller ønsker å la GameManageren kontrollere sending av character oppdateringer, og la den motta oppdateringene fra den tilkoblede host/client, trenger man kun å legge til noen forskjellige Send() og GetInformatio() funksjoner.

All bilde behandling følger et arv hierarki, der man har en ImageObject på toppen. For Statiske bilder som f.eks skal fungere som bakgrunner eller lignende kan man opprette et ImageObject objekt. Man kan sjekke kollisjon mellom ImageObjects, og et ImageObject kan også være usynlig/uten en surface. AnimationObject arver fra ImageObject, og legger til funksjonalitet for å håndtere et sprite ImageObject. Character- og EnemyObject arver begge fra AnimationObject og legger til spesifikk informasjon som som trengs for en player og enemy. EnemyObject arver også fra SimpleAI, som kontrollerer forflyttningen av enemien.





^ Arvhierarki for NPC-Fiender(EnemyObject) og Spillere(CharacterObject)

# Forbedringspotensiale:

Da det ikke var en spesifikk del av oppgaven å lage en enemy med AI, og eventuelt sende informasjon om denne over nettverk, har vi ikke prioritert dette. Vi har likevel påbegynt en simpel enemy klasse og en simpel AI klasse som enemy kan arve fra. Disse to klassene har stort rom for forbedring, da det foreløpig kun er implementert på en så enkel måte som mulig.

Nettverk klassen, incomingMessageManager, vil tilnærmet alltid kunne motta packets. Unntaket er om den første packeten man mottar er mindre enn 4 bytes. I det tilfellet vil MessageManageren ikke kunne sjekke hvordan type packet det er, og den vil bli fjernet. Dette kunne vært løst ved å beholde pakken, for så å legge den sammen med den neste pakken som kommer, men det ble ikke en prioritet å fikse dette da det «aldri» vil skje. Så lenge pakken som kommer er 4 bytes eller større, vil MessageManageren automatisk ta vare på og sette sammen til komplette messages etter hvert som packets kommer.

Med unntak av å passe på at pakkene man mottar stemmer overens med pakker man kan motta, kontrolleres ikke verdiene i pakkene. Dette gir en client muligheten til å «jukse» ved å f.eks sende en høyere velocity enn han skal ha. Her kunne vi ha kontrollert innholdet i pakkene for å se at verdiene stemmer, og eventuelt endret dem slik at ikke en client kan jukse.

Vi har optimalisert mengden data som blir sendt mellom en client og host, men bare til en viss grad. Vi sender oppdateringer om velocity endrer seg, noe som gjør at om man beveger seg med en fast velocity, vil man kun sende en oppdatering når man starter å bevege seg og når man slutter å bevege seg. Det blir også sendt med en posisjon for ekstra kontroll. Ulempen med dette er at om man hopper, vil velocity endres hver oppdatering av spillet, og man ender opp med å sende en oppdatering over nett hver Update(). Dette kunne vært endret ved å sende akselerasjon (run/jump force) når den blir lagt til en character, og la mottakeren av meldingen selv regne ut hva possisjonen blir ved å legge regne ut velocity og legge på gravitasjon.

Om man prøver å starte som client, uten at hosten man vil connecte til har startet, vil enginen henge en stund mens den prøver å connecte, frem til den gir opp. Da vil man ende opp med å starte et nytt game, uten noen informasjon om hosten sine characters eller lignende. Her ville det vært en fordel om det var en definert «time-out» før man gir opp å prøve å connecte, og en melding om dette på menyen. Det vil uansett bli skrevet ut en logg-melding som sier man ikke fikk connected til hosten.

Det bør også nevnes at vi kunne ha brukt GCPtr (Garbage collector pointer) mer for og både sikre oss mot memory leaks, og korte ned på noe kode. Vi har brukt det på en god del dynamisk allokerte objekter og variabler, men ikke alle. Dette skyldes at vi har vent oss til å manuelt rydde opp med delete fra tidligere prosjekter og det ble gjort som en vane sak i enkelte tilfeller. Det ville nok vært en forbedring, først og fremst full sikkerhet mot memory leaks, om alt dynamisk allokert minne brukte GCPtr.

På mottagersiden vil både enemy og players opprettes som characterObjects da de foreløpig kun trenger modellnavn og velocity for å sende updates.

# Ideer vi ikke fikk tid til å implementere:

* Forbedret AI så monsterne kunne virke «Intelligente»
* Støtte for Collision mellom Players og Enemies med per pixel collision
* Implementering av mulighet for å angripe players i tillegg til å kunne redusere HP
* Diverse optimaliseringer og forbedret støtte for Exception-håndtering.
* Sending av character acceleration(force) over net, og la mottakere selv regne ut velocity.

Vi vurderte å implementere egne threads for loading av resources og eventuelt nettverk og avspilling av video på egen thread. Dette var dessverre ikke en del av oppgaven, så vi fikk ikke prioritert det.

Nettverk delen av enginen er laget med tanke på å enkelt kunne utvides til bruke for flere enn 2 spillere. Dette er ikke ferdig implementert da det ikke var en prioritet for innleveringen, men det er ikke mye som skal til for å la hosten ta imot connections fra flere clienter.

# Kilder:

<http://www.devarticles.com/c/a/Cplusplus/A-Simple-Garbage-Collector-for-C-plus-plus/3/>

<http://lazyfoo.net/SDL_tutorials/index.php>

<http://content.gpwiki.org/index.php/SDL:Tutorial:Using_SDL_net>

<http://www.libsdl.org/cgi/docwiki.cgi/>

<http://msdn.microsoft.com/en-US/library/60k1461a%28v=vs.80%29.aspx>

<http://www.boost.org/doc/>

Diverse bruker-spørsmål på <http://stackoverflow.com/> & <http://www.gamedev.net>

Deitel; Deitel - C++ How to Program – 7. Utgave (2009) – ISBN 9780132465403   
  
Foiler og Løsningsforslag

# Copyrights:

Musikk og Karaktergrafikk –rettigheter er eid av Tokyo TV og PLT Enk. Dette spillet er ikke for utgivelse.