# Rapport – Avancerad C++

Isak sondell

## • Problem

Ett 2D-spel i vilket en spelares karaktär och dennes fiender figurerar skall byggas. Dessa fiender skall ha för mål att ta sig förbi karaktären, vilket denne skall förhindra genom att beskjuta dem med sitt vapen.

Spelet kommer att efterlikna en kommandotolk vänd på sin sida, och spelaren kommer att mata in kommandon för att kontrollera spelet och avfyra sitt vapen.

Spelet kommer att utspela sig på användarens hela skärmupplösning, men med ett fönster begränsat till cirka 500x500 pixlar. Spelaren kommer att söka efter fienderna samt flytta på karaktären genom att flytta på fönstret.

## • Design

Spelet, som har dubbats till *”Whitespace Invaders”*, kommer som nämnt utgöra sig för att vara ett konsollfönster.

Spelet skall byggas med hjälp av Simple Fast Media Library 2.3.2 och C++.

SFML har en robust och renderbar text-klass med transform-kababiliteter, denna kommer att stå till grund för det mesta i grafisk väg i *whitespace invaders* istället för texturer. Den grundläggande basklassen kommer representera en sfml-text och all tillhörande funktionalitet som bedöms lämplig, kallad *”Line”.*

*"Line"* kommer att besitta de egenskaper vilka *"Invader", Player", "Trail"* och *"Splosion",* har gemensamt; till exempel tillståndet för huruvida texten följer skärmens förflyttning eller ej, eller att de studsar mot spelskärmens kanter när de rör sig. Dess underklasser kommer däremot att behöva uppdateras på olika sätt.

En styrande hanterare kommer att sköta interaktioner mellan de olika *Line*s som är aktiva på skärmen. Denna kommer även svara för uppdatering och rendering av dessa entiteter. Denna hanterare kommer att följa ett singleton-mönster för att kunna nås av olika klasser under körtid. Exempel på interaktioner som sköts via hanteraren är kollisioner och resulterande explosioner, out-of-bounds-checks, eller poängsättning.

En statisk klass kallad *”LineWriter”* kommer att ansvara för skapandet av de olika entiteter som kommer att agera i spelutrymmet. Denna klass tar med undantag för spelarens karaktär emot *Line*s att efterlikna i skapandet av *Invader*s och *Lazer*s, samt motsvarandens *Splosion*s. Avsikten är att man i källkoden ändrar textens utseende i *Player*-metoden i *LineWriter*, för att uppnå dessa ändringar i alla instanser av *Line* och dess underklasser.

En spelsession kommer att delas upp i *”Screens”*, vilkas egenskaper avgör hur huvudspelet instansierar dessa *Line*s. Exempelvis kommer *MainScreen* att under krympande intervall att instansiera *Invader* klassen, vilka kostar spelaren ett liv om de ej skjuts ner innan de når underkanten på spelarens skärm. Dessa *Screen*s kommer att avlösas i en cyklisk ordning som sköts av en separat hanterare och stegar igenom följden, detta för att kunna lägga till eller ta bort *Screen*s allt eftersom de blir färdigskrivna eller förlegade.

# • Analys

Det är uppenbart att implementationen lider av strukturella brister, spelets exekvering träder fram och tillbaka mellan sina hanterare och entiteter flera gånger per enskild entitet och uppdatering, på grund av hur mycket av funktionaliteten som implementeras i en annan klass än den borde.

Grundläggande funktionalitet som att till exempel återställa spelet är inte centrala, utan har kopierats ut i diverse funktioner och således finns ingen särskild kontroll över dem.

Samma problem återfinns och återkommer i flera klasser, i synnerhet ObjectHandler är en otymplig klass med för röriga och för stora metoder, detta ledde exempelvis till stora svårigheter när en minnesläcka upptäcktes som skedde någonstans under ObjectHandler's uppdateringsmetod, denna läcka blev onödigt svår att hitta.

Const-inkorrekthet råder i allra högsta grad, och många gånger av ren lathet. Basklassen *Line* i synnerhet borde ha haft fler mer utbyggda och const-korrekta metoder för att behandla sina medlemsobjekt, väldigt många problem under utvecklingen har härstammat från denna klass, och kunde ha förebyggts;

GamePlay-designen bör inte tas upp i för stor utsträckning men det borde antagligen nämnas att om en tydligare designriktning tagits från första början kunde de första UML-skisserna varit mer applicerbara, men källkoden kunde också ha varit mycket mer mottaglig för det förändringar som var sannolika, det finns t.ex. medlemsvariabler som roterar spelplanen till en mer *"konsoll-aktig"* horisontell sådan, men endast vissa delar av koden stödjer den.

Metoderna och klasserna som sköter explosioner och *trails* är relativt lättskötta, även om de skulle kunna skötas från en Effekt-klass. Men de expanderar spelarens upplevelse avsevärt per investerad utvecklingstid.

Soundtrack, spelljud, high-scores, och faktisk score över huvud taget, lyser med sin frånvaro; de föll offer för felsökning, tillsammans med speldesignen till att börja med. Måhända kunde originaltanken med transparens mellan *Splash, Main Menu, Main Game, Game Over, Credits, etc,* fungerat med en robustare implementation.

# • Körexempel

Spelet startas upp till vad som spelaren förhoppningsvis begriper vara en usel avbild av ett konsollfönster eller en kommandotolk **(se figur 2).**

Med denna vågade förhoppning i ryggen antas då även att spelaren kommer att försöka skriva in ett kommando.

Detta matar in knapptryckningar i *Player-klassens EnterText*-metod, vilken ritar ut texten framför spelaren, det så kallade kommandot.

Varpå spelaren trycker på Enter matchas kommandot mot kända kommandon i den statiska klassen *Commands*, som sedan skapar entiteter utifrån dem.

Alla de entiteter som skapas görs så av den statiska klassen *LineWriter*, varpå de matas in i en passande *std::vector* som bibehålls av *ObjectHandler*.

Felmeddelanden och hjälpmeddelanden ritas ut på skärmen och flyttar sedan fönstret "lika många rader", spelarens position fortfarande är central **(se figur 3).**

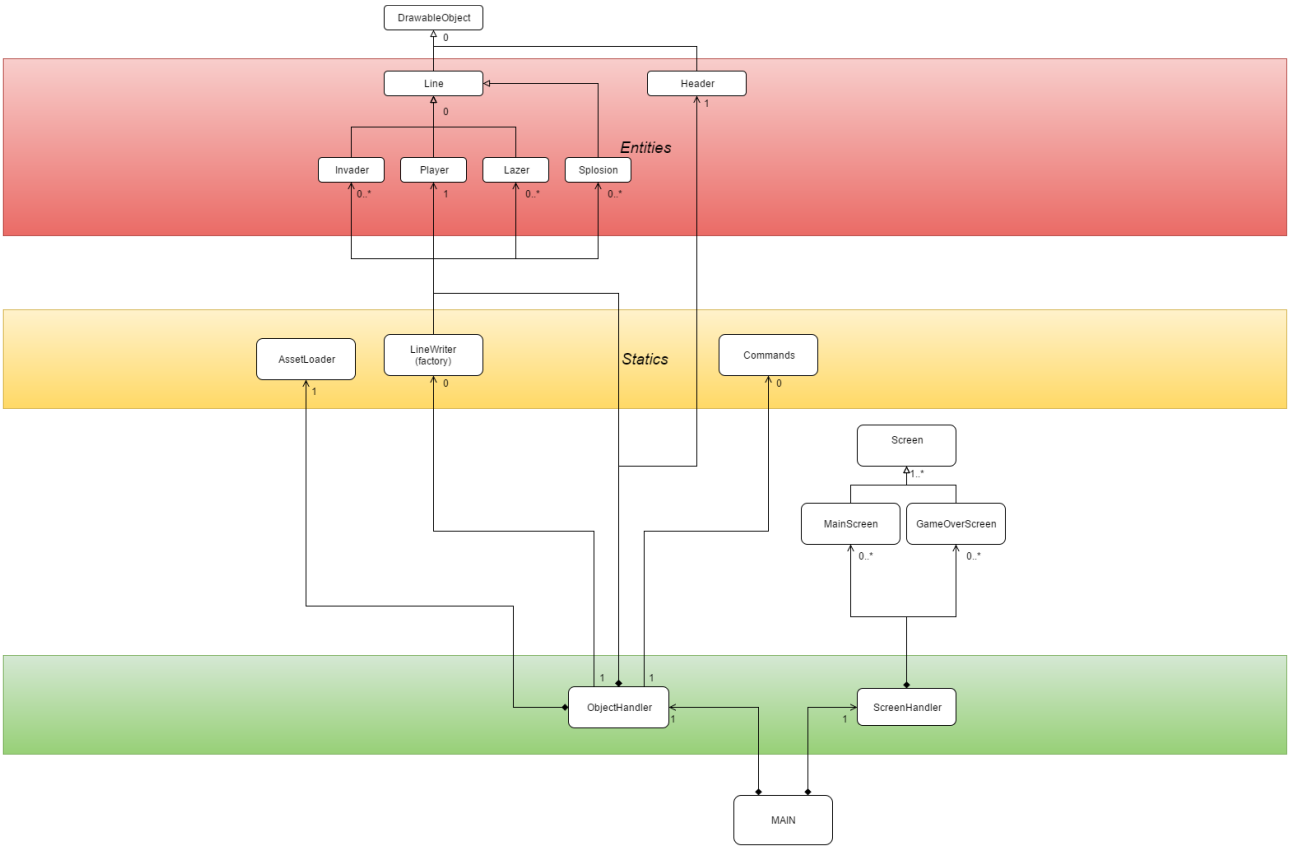
Det stora antagandet här är att spelaren skall upptäcka fönstrets rörelse, och sedan själv vilja flytta det med hjälp av musen och även upptäcka spelets huvudmekanik, vare sig spelaren upptäcker detta eller ej förväntas denne trycka på alt+f4 i besvikelse, men vid den eventualitet att hen är mer ihärdig än förutsett kan även någon liknelse vid ett datorspel därefter spelas.

Om *Commands* upptäcker kommandot ”newgame” så börjar *ScreenHandler* uppdatera mot *MainScreen*, vilken instantierar *Invader*s i ett krympande intervall, spelets mål är att med kommandot ”lazer” eller ”pew” instantiera *Lazer*-entiteter som vid kollision med *Invader*s resulterar i *Splosion*s, och på så vis hindra dem från att nå Windows-miljöns undre kant (märk att det inte är fråga om spelfönstrets undre kant) **(se figur 4).**

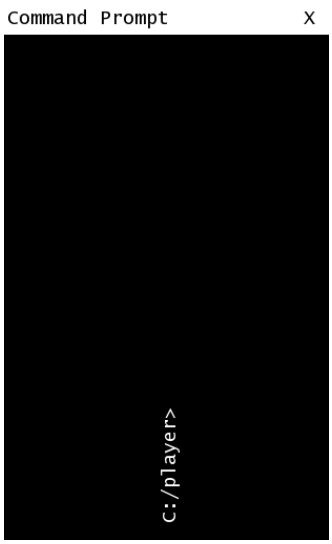
*ObjectHandler* som sköter uppdateringen av alla *Line*-deriverade entiteter ser också till att dra av ett liv från spelaren var gång, och om det upptäcks att spelaren har slut på liv ombedes *ScreenHandler* att byta *Screen*, förhoppningsvis till en som gör det klart för spelaren att hen har förlorat **(se figur 5)**.
Spelet startas om genom mata in kommandot ”newgame”, vilket kan göras när som helst.

När en *Line*-entitet skall förstöras sätts flaggas denne för att förgöras med en medlemsboolean, vilken kontrolleras i nästa uppdatering av *ObjectHandler* som sedan ersätter entiteten med en *Splosion*-entitet per bokstav i ursprungsentitetens text.
När spelets tillstånd skiftar från en *Screen* till en annan, eller när helst en *Invader* tar ett av spelarens liv, så flaggas även samtliga *Invaders* och *Lazers på detta vis*.

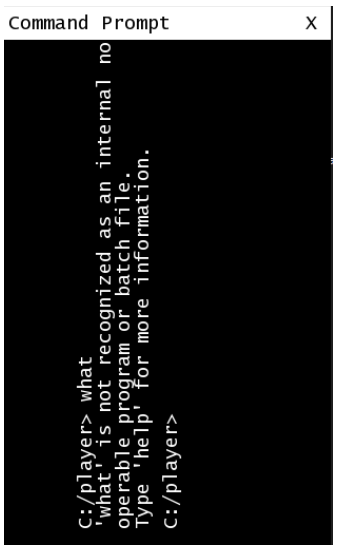
# Figurer



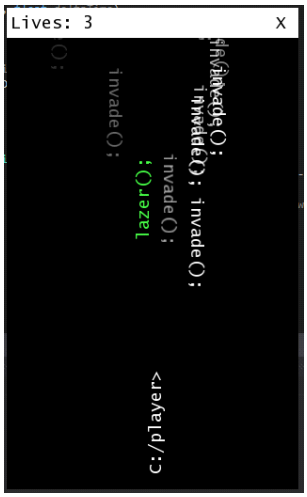

*Figur 1, UML-Diagram*



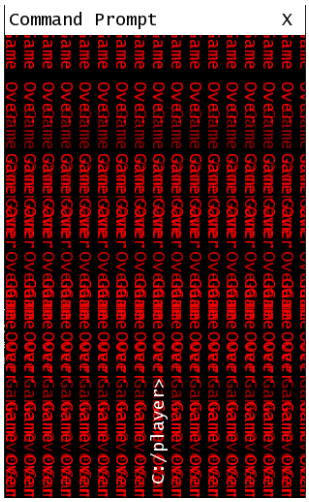

*Figur 2, spelets första och sannolikt sista intryck.*




*Figur 3, felmeddelandet flyttar skärmen.*




*Figur 4, invadörer och en laser..*




*Figur 5, Game Over.*