# CPROG Rapport för Programmeringsprojektet

[Gruppnummer: 34]

[Gruppmedlemmar: Viktoria Athanasopoulou 951224-4428]

*Skriv en kortfattad instruktion för hur programmeringsprojektet skall byggas och testas, vilka krav som måste vara uppfyllda, sökvägar till resursfiler(bildfiler/ljudfiler/typsnitt mm), samt vad en spelare förväntas göra i spelet, hur figurernas rörelser kontrolleras, mm.*

*Om avsteg gjorts från kraven på Filstruktur, så måste också detta motiveras och beskrivas i rapporten.*

*Fyll i 'check-listan', så att du visar att du tagit hänsyn till respektive krav, skriv också en kort kommentar om på vilket sätt du/gruppen anser att kravet tillgodosetts, och/eller var i koden kravet uppfylls.*

*Den ifyllda Rapportmallen lämnas in tillsammans med Programmeringsprojektet*. Spara rapporten som en PDF med namnet CPROG\_RAPPORT\_GRUPP\_NR.pdf (där NR är gruppnumret).

## Beskrivning

Denna spelmotor består av 2 typer av projektfiler: Backendfiler och komponenter. Vi har följt den rekommenderade filstrukturen men gjort undermappar för dessa 2 typer under /src för en tydligare struktur.

Backendfilerna är **System** och **Session** som ser till att sessionen går igenom alla komponenter och anropar deras virtuella medlemsfunktion tick() som styr vad de gör. Component är en abstrakt klass som också ligger i backend-mappen och är därifrån alla komponenter ärver från:

* **SpaceShip**. Ritar upp ett spaceship längst ned på fönstret som kan skjuta bullets uppåt genom att trycka på space, samt röra sig på skärmen med hjälp av pilarna.
* **Goblin**: Aliens som regnar ned från toppen av fönstret, och genereras på random x-positioner.
* **Bullet**. Kan skjutas från en annan komponent uppåt tills den träffar något.
* **Warrior**. Gubbe som kan röra sig över skärmen med hjälp av pilarna, samt skjuta med space.

Man kan skapa olika spel genom att ändra i Main.cpp för att skapa och lägga till komponenterna man vill ha.

Till exempel för att skapa Space invaders skapar man pekare till SpaceShip och Goblin, lägger till dem i sessionen och kör genom att kalla på add() och run() metoderna hos cwing::Session::ses. Detta exemplet är vad vi har gjort i den bifogade Main-klassen. Vi har redan kompilerat programmet så det är nu en körbar exe-fil i build/debug/play i projektmappen.

## Instruktion för att bygga och testa

Denna spelmotor kan byggas i en kompileringsmiljö för C++ samt ha SDL2 och dess tilläggsbibliotek.

Vi har valt att bygga programmeringsprojektet genom make i VSCode, i kombination med GNU:s kompilator g++ (installerat genom MSYS2 MinGW), samt SDL2 (och dess tilläggsbibliotek). När det gäller debugging har vi använt oss av en GDB/LLDB-konfiguration.

Resursfiler ligger i /resources där det finns en mapp per kategori (fonts, images, sounds).

## Krav på den Generella Delen(Spelmotorn)

* 1. [ Ja ] Programmet kodas i C++ och grafikbiblioteket SDL2 används.
  2. [ Ja ] Objektorienterad programmering används, dvs. programmet är uppdelat i klasser och använder av oo-tekniker som inkapsling, arv och polymorfism. ***Kommentar****: Vi har en rot-klass för komponenter med virtuella funktioner, har kapslat in privata variabler.*
  3. [ Ja ] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.  
     ***Kommentar****: Ja, tillämpningsprogrammeraren är skyddad för att det går inte att skapa object utav Component som är en polymorf klass*.
  4. [ Ja ] Det finns en gemensam basklass för alla figurer(rörliga objekt), och denna basklass är förberedd för att vara en rotklass i en klasshierarki.
  5. [ Ja ] Inkapsling: datamedlemmar är privata, om inte ange skäl.
  6. [ Ja/Nej/Delvis ] Det finns inte något minnesläckage, dvs. jag har testat och sett till att dynamiskt allokerat minne städas bort.  
     *Kommentar: skriv din kommentar här…*
  7. [ Ja ] Spelmotorn kan ta emot input (tangentbordshändelser, mushändelser) och reagera på dem enligt tillämpningsprogrammets önskemål, eller vidarebefordra dem till tillämpningens objekt.
  8. [ Ja ] Spelmotorn har stöd för kollisionsdetektering: dvs. det går att kolla om en Sprite har kolliderat med en annan Sprite. ***Kommentar****:*

*Vi har en metod i Session som går igenom alla komponenter och kollar detta. Varje komponent har en egen tillämpning som returnerar falskt om de vill bli borttagna vid kollision, eller sant om de vill vara kvar i sessionen.*

* 1. [ Ja ] Programmet är kompilerbart och körbart på en dator under både Mac, Linux och MS Windows (alltså inga plattformspecifika konstruktioner) med SDL 2 och SDL2\_ttf, SDL2\_image och SDL2\_mixer.

## Krav på den Specifika Delen(Spelet som använder sig av Spelmotorn)

* 1. [ Ja ] Spelet simulerar en värld som innehåller olika typer av visuella objekt. Objekten har olika beteenden och rör sig i världen och agerar på olika sätt när de möter andra objekt.  
     ***Kommentar****:*

*Ja, vårt spel är Space invaders där det regnar ned aliens från toppen av skärmen. Dessa kan man skjuta med sitt spaceship, när kulan kolliderar med en goblin försvinner både kulan och goblin från sessionen.*

* 1. [ Ja ] Det finns minst två olika typer av objekt, och det finns flera instanser av minst ett av dessa objekt.
  2. [ Ja ] Figurerna kan röra sig över skärmen.   
     ***Kommentar****: Goblin regnar automatiskt, men SpaceShip kan röra sig med hjälp av piltangenterna. Samt bullets skjuts när man trycker på space.*
  3. [ Ja ] Världen (spelplanen) är tillräckligt stor för att den som spelar skall uppleva att figurerna förflyttar sig i världen.  
     ***Kommentar****: Eftersom goblins regnar nedåt upplevs värden stor.*
  4. [ Ja ] En spelare kan styra en figur, med tangentbordet eller med musen.
  5. [ Ja ] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.  
     Kommentar: Bullets och goblins försvinner från sessionen, men inte spaceship då de har olika implementationer av den metoden.