CPROG Rapport för Programmeringsprojektet

[Gruppnummer: 1]
[Gruppmedlemmar: Hanna Arrhenius 20001020-8122, Louis Guerpillon 20010510-7577]

1. Beskrivning

Spelmotorn har byggts upp i Visual Studios Code med SDL2 där motorn är skapad för att användaren ska kunna göra sitt eget co-op-spel. Hierarkin och diverse data lagrade på motorklasserna finns där för att underlätta detta, exempelvis int-variablen “playerNumber”, som finns för att tilldela vilken av spelarna ett objekt tillhör (om denne så önskar). Exempelvis skulle en programmerare kunna använda sig av playerNumber samt dess change- och get-funktioner för att tilldela ett vapen, skada, eller poäng till den önskade spelaren.

Spelet som har skapats med hjälp av motorn har därför endast två spelare och inga fiender eftersom spelarna kör mot varandra. Det huvudsakliga målet med spelet är att det ska simulera en värld med två olika skepp i rymden som ska eliminera den andra där man ska träffa den andra spelaren med skott för att den ska ta skada. Den som först lyckas göra skada på den andra spelaren vinner. Varje spelare har ett varsitt rymdskepp som den kontrollerar via tangentbordet och det spelas därför via LAN, dvs via ett lokalt nätverk. Spelarna behöver befinna sig vid samma tangentbord för att kunna spela tillsammans och varje spelare har olika tangentbordsknappar som de kontrollerar. Spelare A har A och D för att kunna röra sig fram och tillbaka på skärmen och C för att kunna skjuta skott på spelare B. Spelare B har J och L för rörelse samt M för att kunna skjuta på spelare A. Motorn är uppbyggd för att kunna skapa fler spelare än två vilket gör att användaren tekniskt sett även kan ha fyra spelare som kör mot varandra men för detta spel har vi endast två spelare. Rymdskeppen i spelet kontrolleras endast via tangentbordet för att underlätta samspelet mellan spelarna eftersom det endast finns en mus för varje dator men flera knappar på ett tangentbord. När en spelare skjuter mot den andra spelaren tar den skada ifall denna träffas av skottet(bullet) och varje spelare har 10 liv(health). Varje skott som träffar en spelare tar ett liv vilket gör att varje spelare behöver träffas av 10 skott för att förlora. Om en spelare har blivit av med sina 10 liv försvinner denna och den andra spelaren vinner.

2. Instruktion för att bygga och testa

I resursfilerna befinner sig “resources” mappen där alla components som programmet använder sig av är lagrade. För att förenkla processen av att byta ut de sprites och diverse visuella komponenter som programmeraren önskar visa upp använder programmet sig av Constants.h för att smidigt underlätta denna process. Vill programmeraren skapa en subklass till ett objekt (i detta fall döpt “Component”), behöver denne enbart ange en förenklad sökväg till bilden således denna finns i resursmappen. Ett exempel på detta är vore att initiera en komponent på detta vis:

"images/exempel.png”

Spelet

För att bygga ett spel med hjälp av motorn skapar man först en ny fil, i detta testfall är filen döpt efter spelet (SpaceDuel.cpp). I den finner man en main-funktion som är den funktionen som startar igång det byggda spelet. I vår SpaceDuel.cpp fil inkluderas spelunika klasser, som till exempel Ship.h, som är en subklass till den generella "Component"-klassen. Efter att alla önskade spelklasser byggts färdigt är det dags för programmeraren att lägga till dessa till en session av spelmotorn. För att göra det implementerar man ett namespace vid namnet GameEngine som låter en nå alla spelmotorklasser direkt i main. För att lägga till ett objekt till spelet kan man använda sig av engine, som tack vare dess add()-metod låter programmerare lägga till objekt av spelklasserna direkt in till spelet. När man är nöjd med vad som adderats startar man igång motorn med engine.run(), som skrivs näst längst ner i main-metoden, precis över main-metodens "return 0;".

Då motorn är anspassad för att skapa multiplayer-spel kräver den att minst två spelare är initierade innan run() får köras. Man initierar spelare med hjälp av engine's InitializePlayers()-funktion. I vårt fall initieras två instanser av spelklassen "Ship", men eftersom InitializePlayers() tar en generell komponent som parameter är denna anpassad för alla nya sorters spelklasser som programmeraren önskar initiera.

3. Krav på den Generella Delen(Spelmotorn)

3.1. [Ja/Nej/Delvis] Programmet kodas i C++ och grafikbiblioteket SDL2 används.
Kommentar: Ja, programmet kodas i C++ samt alla klasser som behöver det använder sig av SDL2 vilket i detta fall endast är spelmotorns klasser.

3.2. [Ja/Nej/Delvis] Objektorienterad programmering används, dvs. programmet är uppdelat i klasser och använder av oo-tekniker som inkapsling, arv och polymorfism.
Kommentar: Ja, programmet är uppdelat i flera klasser som tillsammans skapar den generella spelmotorn. Motorklassen Component har flera subklasser som tillsammans formar spelet.

3.3. [Ja/Nej/Delvis] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.
Kommentar: Ja, både tilldelningsoperatorn och kopieringskonstruktorn är förbjudna i Component-klassen.

3.4. [Ja/Nej/Delvis] Det finns en gemensam basklass för alla figurer(rörliga objekt), och denna basklass är förberedd för att vara en rotklass i en klasshierarki.
Kommentar: Ja, denna klass heter Component. Alla spelkomponenter är subklasser av Component.

3.5. [Ja/Nej/Delvis] Inkapsling: datamedlemmar är privata, om inte ange skäl.
Kommentar: Ja, alla som kan vara privata är det förutom vissa som är protected för

att tillåta subklasser att nå dessa.

3.6. [Ja/Nej/Delvis] Det finns inte något minnesläckage, dvs. jag har testat och sett till att dynamiskt allokerat minne städas bort.
Kommentar: Ja, minne som allokeras städas bort genom destrukorn i Component vilket underlättas med en for-loop som itererar genom remove-vektorn.

3.7. [Ja/Nej/Delvis] Spelmotorn kan ta emot input (tangentbordshändelser, mushändelser) och reagera på dem enligt tillämpningsprogrammets önskemål, eller vidarebefordra dem till tillämpningens objekt.
Kommentar: Ja, i vår GameEngine stödjs inputhantering. Denna input kan skickas till önskat objekt, och "Component"'s funktioner som stödjer input kan överlagras för att komma på ny mekanik och logik från angiven input.

3.8. [Ja/Nej/Delvis] Spelmotorn har stöd för kollisionsdetektering: dvs. det går att kolla om en Sprite har kolliderat med en annan Sprite.
Kommentar: Ja, varje komponent som skapas i spelvärlden har en funktion som kollar och tar sedan en annan komponent som parameter och därefter returnerar ett booleskt värde som kommunicerar vare sig komponenten i frågan kolliderar med komponent i parametern.

3.9. [Ja/Nej/Delvis] 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.
Kommentar: Ja.

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

4.1. [Ja/Nej/Delvis] 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. Det finns två objekt "player(ship)" som har en texture i sin konstruktor som gör att en bild kan visas på objektet. Objekten kallas ship och använder sig av tangentbordsinput för att kontrolleras av användaren. Objektet använder sig av collision för att se till att den tar skada när det ena objektet skjuter en bullet på den. Detta är då kollisionen i koden som gör att objekten agerar på olika sätt vid möte med ett annat objekt, koden ser då till att när ship träffas av bullet tar den skada och dess health sänks. Bullets instansieras av Ship och beter sig olika beroende på vilken spelare som avfyrat den, exempelvis kan den enbart kollidera med motståndarens skepp. När ship träffats av 10 bullets kommer denna att tas bort(delete) vilket gör att den försvinner från spelvärlden.

4.2. [Ja/Nej/Delvis] Det finns minst två olika typer av objekt, och det finns flera instanser av minst ett av dessa objekt.
Kommentar: Ja, här har vi både component bullet och ship som collidar, det finns

även två instanser av ett ship som beter sig på olika sätt. Flera bullets kan instansieras samtidigt och dessa avfyras av båda spelarna.

4.3. [Ja/Nej/Delvis] Figurerna kan röra sig över skärmen.
Kommentar: Ja. Ship rör sig fram och tillbaka på skärmen och har begränsad rörelse, spelare A kan endast röra sitt skepp högst upp på skärmen som i en viss “bana” då den endast kan röra sig fram och tillbaka med ett stopp vid slutet av skärmen för att den inte ska gå utanför fönstret. Samma gäller för spelare B och dess skepp.

4.4. [Ja/Nej/Delvis] Världen (spelplanen) är tillräckligt stor för att den som spelar skall uppleva att figurerna förflyttar sig i världen.
Kommentar: Ja. Spelmotorn kan lätt anpassas för olika typer av storlekar på fönstret baserat på vad för spel som ska skapas, spelet vi skapar är 1024x576 i fönstret men vid andra/nya spel kan detta ändras.

4.5. [Ja/Nej/Delvis] En spelare kan styra en figur, med tangentbordet eller med musen.
Kommentar: Ja. Vi har skapat ett spel där man spelar två stycken genom att man delar på tangentbordet och varje spelare använder olika tangenter.

4.6. [Ja/Nej/Delvis] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.
Kommentar: Ja. När en bullet träffar ett ship/spelare tar denna skada i form av health, när den har tagit en viss mängd skada kommer det ena skeppet tas bort från spelplanen och den andra spelaren vinner. Resultatet av ett möte beror på vad för objekt som möts, exempelvis kolliderar inte två bullets i varandra, utan enbart med dess avfyrares fiende.