CPROG Rapport för Programmeringsprojektet

Gruppnummer: 001
Gruppmedlemmar: Fredrik Halling → (19940727)
Jameel Saeed → (19970228)
Rasmus Rodriguez → (19940613)

1. Beskrivning

Vi har skapat en enkel space shooter som går ut på att spelaren ska skjuta ned anfallare som kommer in från toppen av rutan och som försöker ta sig ned till botten av rutan. Om en anfallare kolliderar med botten av rutan eller spelaren så förlorar man och spelet stängs ned. Så länge en spelare hinner skjuta ned anfallare som spawnar så fortsätter spelet.

2. Instruktion för att bygga och testa

Man behöver anpassa Makefile för ens OS/System, vår Makefile är anpassad för MacOS Big Sur (version 11.5.2 och senare) med kompilatorn Apple Clang (version 12.0.5) installerat.

Bibliotek som är nödvändiga för att kompilera programmet är SDL2, SDL2_image och SDL2_mixer. Dessa behöver vara installerade innan spelet kompileras och körs. Dessa hämtas och installeras lättast för MacOS med hjälp av Homebrew.

För att kompilera programmet anropar man kommandot `make` i projektets mapp via Terminal. Därefter kan man exekvera programmet *play* som under kompilering lagts under mappen project/`build/debug/`.

När spelet startar ska spelaren skjuta ned alla anfallande karaktärer från att ta sig ned till botten av rutan. Detta görs genom att flytta spelaren i sidled med hjälp av vänster, respektive höger piltangent och skjuta genom att trycka ned tangenten för mellanslag. Detta görs tills dess att en anfallare lyckas kollidera med botten av rutan eller med spelaren, varpå spelet avslutas. För att avsluta spelet tidigare kryssar man på enklaste sätt rutan.

För att kontrollera spelet mot eventuella minnesläckage kan man på MacOS använda Leaks genom att i Terminal starta spelet genom Leaks med följande kommando: `leaks --atExit -- ./build/debug/play`.

3. Krav på den Generella Delen(Spelmotorn)

3.1. [Ja/Nej/Delvis] Programmet kodas i C++ och grafikbiblioteket SDL2 används.

Kommentar: Ja.

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. Samtliga grafiska objekt som ritas ut på skärmen är subklasser till basklassen Component. Vidare finns en Main-klass som syftar till att initiera ett objekt av klassen Session. Det är i sessions run-metod som spelet existerar under körning.

3.3. [Ja/Nej/Delvis] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.

Kommentar: Ja. Exempelvis genom att tillämpningsprogrammeraren tvingas skapa nya instanser av Player, Enemy och Bullet på stacken genom deras metoder getInstance() som returnerar en ny instans av respektive klass. Detta så att programmeraren inte av misstag eller medvetet kan skapa nya objekt av respektive klass på heapen.

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. Component är en basklass för samtliga rörliga objekt vi använder oss genom spelet. Det går inte att skapa objekt av Component, endast av dess subklasser.

3.5. [Ja/Nej/Delvis] Inkapsling: datamedlemmar är privata, om inte ange skäl.

Kommentar: Ja. Samtliga datamedlemmar för samtliga klasser är privata med undantaget för Rect i Component som är protected. Detta eftersom att Rect behöver kommas åt och sättas genom super-anrop vid konstruering av instanser av subklasser till Component.

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. Detta har testas genom att köra spelet genom programmet Leaks för MacOS, kommandot för att göra detta är: `leaks --atExit -- ./build/debug/play`. Grafiska objekt städas bort när de inte längre är relevanta för spelet. Dessa kontrolleras genom att beräkna objektets relation till systemfönstret.

När spelet avslutas kallas destruktorn av Session där samtliga grafiska objekt som har instansierats städas bort. Således undviks minnesläckage.

```
Session::~Session()
   {
       removed.clear();
       added.clear();
       for (std::vector<Component*>::iterator iter = comps.begin();
             iter !=  comps.end();)
       {
           delete *iter;
           iter++;
       }
   }
```

(rad 151, Session.cpp)

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. Vi har valt att implementera SDL_KEYDOWN i vår session-loop som hanteras ett SDL-event. Eventet skickas vidare till samtliga komponenter som i sin tur hanterar eventet och implementerar beteende eller inte för tangenten som trycks ned.

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. För varje varv i spelmotorns huvudloop kontrolleras det om komponenter har kolliderat med varandra. Kontrollen sker via en boolsk hjälpmetod som heter `detectCollision`. Om en kollision har inträffat placeras de kolliderande komponenterna i en temporär vektor `removed`, varefter `removed` itereras över och alla kolliderande komponenter som finns med där tas bort från vektorn `comps`.

```
bool Session::detectCollision(Component* a, Component* b)
   {
       if(a->getRect().x < b->getRect().x + b->getRect().w
       && a->getRect().x + a->getRect().w > b->getRect().x
       && a->getRect().y < b->getRect().y + b->getRect().h
       && a->getRect().y + a->getRect().h > b->getRect().y)
       {
           return true;
       }
       return false;
   }
```

(rad 139, Session.cpp)

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 SDL2 och SDL2_ttf, SDL2_image och SDL2_mixer.

Kommentar: Ja. Vid utvecklandet av spelet har både macOS Big Sur och Linux Debian 11 använts. Tyvärr har vi inte haft möjlighet att testa programmet på en dator med Windows som operativsystem, då vi inte har haft tillgång till en sådan. Eftersom att spelet går att kompilera och köra på både macOS och Linux så förutsätter vi det även fungerar på Windows.

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. Samtliga grafiska objekt ärver från basklassen Component. Subklasserna förhåller till varandra på olika sätt. När en Bullet kolliderar med en Enemy försvinner dessa från skärmen och objekten städas bort. Däremot avslutas spelet om Player-objektet kolliderar med ett Enemy objekt.

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. Player-objektet är unikt i spelimplementationen. Enemy-objektet skapas i förhållande till en gameCounter (int) variabel som räknas upp för varje varv i huvudloopen. Bullet objekt skapas på begäran av spelaren när hen trycker på mellanslag. Det kan existera många Bullet och Enemy objekt samtidigt. Båda städas bort när dessa inte längre är relevanta.

4.3. [Ja/Nej/Delvis] Figurerna kan röra sig över skärmen.

Kommentar: Ja. Player objektet styrs av spelaren med hjälp av piltangenterna. För att speltemat ska vara intakt kan spelaren endast röra sig i sidled. Enemy rör sig i konstant hastighet nedåt över skärmen. Bullet rör sig i konstant hastighet uppåt över skärmen.

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. När Enemy och Bullet objekt skapas visas dessa under en relevant tidsperiod. Under perioden som ett Enemy-objekt är relevant tvingas spelaren eliminera objektet för att spelet inte ska avslutas. Bullet-objekt, som till sin natur är snabba, färdas i hög hastighet över skärmen.

4.5. [Ja/Nej/Delvis] En spelare kan styra en figur, med tangentbordet eller med musen.

Kommentar: Ja. Spelaren kontrollerar Player-objektet med höger och vänster piltangent. För att skjuta trycker man ned mellanslag. Anrop från musen är inte implementerade.

4.6. [Ja/Nej/Delvis] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.

Kommentar: Ja. När två komponenter kollideras uppstår en kollision. Båda objekten placeras i en temporär vektorn `removed`. Därefter plockas dessa objekt bort från vektorn `comps` och kommer således inte att ritas ut på skärmen. Vidare städas de kolliderade objekten bort för att förhindra minnesläckage.