CPROG Rapport för Programmeringsprojektet

[Gruppnummer: **35**]
[Gruppmedlemmar: **Jacob Andersson 19950524-3916**

1. Beskrivning

Spelet är ett väldig simpelt 2player spel som går ut på att den ena spelare (Skytten, i mitt spel figuren Link) ska skjuta pilar på den andra figuren(Zelda i mitt spel). Om Zelda träffas av en pil så dör hon och spelet är över. Zelda ska undivka pilarna genom att trycka på space knappen för att hoppa. Spelare 1 skjuter pilar genom att trycka på Shoot knappen. För varje skott som avfyras så räknas poängen upp och den slutgiltiga poängen är Zeldas slutpoäng.

2. Instruktion för att bygga och testa

För att .exe filen i build/debug/ ska kunna köras så måste alla SDL dll finnas i samma mapp som .exe filen. Huvudprogrammet finns i cpp filed main.cpp. Resursfiler finns i resources mappen och för körning av programmet så måste konstanten gResPath i constants.h bytas ut mot lokala sökvägen till mappen. Det finns även andra konstanter i constants.h som påverkar storleken på spelet samt fysiska element som hastighet på projektiler och hastighet på hopp. Undvik att ändra på storlekskonstanterna då detta påverkar hela spelet.

3. Krav på den Generella Delen(Spelmotorn)

3.1. [**Ja**/Nej/Delvis] Programmet kodas i C++ och grafikbiblioteket SDL2 används.
Kommentar: **Spelmotorn använder sig enbart utav SDL komponenter för att grafiskt visa spelet. Koden är skriven i C++ och har skrivits i Visual Studio**

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: **Alla komponenter som ritas upp på skärmen ärver av en klass som heter Sprite. Sub-komponenterna gör egna implementationer av metoder som tick() och draw(). Alla metoder i mina klasser är privata så länge de inte behöver vara publika. De delar av programmet som jag känner inte riktigt håller en bra OO-standard är mina globala variabler såsom System, GameEngine och Game. De två förstnämnda gjorde jag då jag förstod det som att det var så man designade spel med den här tekniken. Dock är jag inte så nöjd med min globala Game variabel, anledningen till att jag gjorde den var för att kunna hantera olika states i mitt spel. Game klassen har en metod som heter update() som anropas varje varv i spelloopen och uppdaterar spelets tillstånd beroende på vad som hänt i spelet.**

3.3. [**Ja**/Nej/Delvis] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.
Kommentar: **Alla klasser som går att instansiera har en protected konstruktor och en statisk get_instance metod, som returnerar en pekare till objektet som skapas. Jag har även gjort copy konstruktor och tildelningsoperatorn privat och =delete så att de inte är tillgängliga för subklasser.**

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: **Denna klass heter Sprite och går inte att instansiera själv utan det måste göras subklasser som ärver av den. Den har metoder som tick() och draw() som är virtuella och måste implementeras av subklasser. Dessa metoder används i spelloopen.**

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

3.6. [**Ja**/Nej/Delvis] Det finns inte något minnesläckage, dvs. jag har testat och sett till att dynamiskt allokerat minne städas bort.
Kommentar:

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: **Spelmotorn lyssnar på alla tangentbordshändelser varje varv i loopen och skickar ett keyboardstate till varje sprite objekt varje varv. Det är sedan subklassernas implementationer som bestämmer hur de ska reagera på dessa händelser. Samma gäller även för mushändelser.**

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: **I headerfilen Maths finns 2 metoder för killisionsdetektering. En för den omgivande SDL_rect och en för pixelcollision. Vilken version man vill använda kan man bestämma i spelmotorn. I det medskickade spelet är pixelcollision implementerad men jag valde att behålla SDL_rect detektionen ifall det skulle vara några problem med den andra.**

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:

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: **Spelet utspelar sig i en 2d värld där projektiler färdas längst skärmen och vid kollision med en annan Sprite avslutar spelet.**

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: **Det finns en bågskytt som skjuter pilar och det kan finnas flera pilar i programmet samtidigt. Det finns även 2 spelkaraktärer, en som kan skjuta pilar och en som kan hoppa.**

4.3. [**Ja**/Nej/Delvis] Figurerna kan röra sig över skärmen.
Kommentar: **Pilarna rör sig horisontellt över skärmen och spelfiguren Zelda rör sig vertikalt genom att hoppa.**

4.4. [Ja/Nej/**Delvis**] Världen (spelplanen) är tillräckligt stor för att den som spelar skall uppleva att figurerna förflyttar sig i världen.
Kommentar: **Spelet utspelar sig i en stillastående värld, dvs dem förflyttar sig inte i världen genom att röra sig horisontellt. Dock hoppar ena figuren vertikalt och färdas på så sätt genom spelplanen.**

4.5. [**Ja**/Nej/Delvis] En spelare kan styra en figur, med tangentbordet eller med musen.
Kommentar: **Ena spelaren skjuter med musen genom att trycka på en knapp, den andre spelaren hoppar genom att trycka på ”space” knappen på tangentbordet.**

4.6. [**Ja**/Nej/Delvis] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.
Kommentar: **Om den ena figuren träffas av en pil så försvinner den och spelet går till ett annorlunda ”state”. Där finns det möjlighet för spelaren att starta om spelet genom att trycka på restartknappen.**