CPROG Rapport för Programmeringsprojektet

[Gruppnummer: 13]
[Gruppmedlemmar: Wilhelm Durelius 990106, Joshua Kostian 970220

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).

1. Beskrivning

Spel 1: j - floppy box

I detta spel använder man enbart space bar för att kontrollera karaktären, som då hoppar. Målet är att hoppa igenom så många rör som möjligt utan att röra någon av dem, eller komma utanför rutan. Spelet startas om vid 'död'.

Spel 2: w - scuffed snek

I detta spel styr man antingen med WASD, eller piltangenterna. Man ska plocka på sig mat (orbs) och varje gång man äter en orb får man en till tail. Åker ormen in i sig själv så dör ormen. Klicka då på space för att starta om. Poängen står längst upp i vänstra hörnet.

2. Instruktion för att bygga och testa

- SDL3, SDL3_image och SDL3_ttf behöver vara installerat och finnas i din path
- Du behöver en C++ kompilator (g++-14 för Mac, g++ för Linux och Windows)
- Spelen kompileras och körs genom antingen "make w" eller "make j" från projektroten.
  Make j kompilerar och kör spelet 'floppy box' som Joshua skrivit.
  Make w kompilerar och kör spelet 'scuffed snek' som Wilhelm skrivit.

Alla sökvägar till resources finns i resources mapparna, där de är definierade med relativa sökvägar

3. Krav på den Generella Delen(Spelmotorn)

3.1. [**Ja**/Nej/Delvis] Programmet kodas i C++ och grafikbiblioteket SDL används.
Kommentar: Ja, enbart C++ och SDL används.

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 klasser och har programmerats på ett OOP-sätt.

3.3. [**Ja**/Nej/Delvis] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.
Kommentar: Shared pointer hanterar värdesemantiken åt oss. Varje instans av klasserna kan enbart skapas dynamiskt via en statisk 'make' metod, som då returnerar en shared pointer. Alla polymorfa klasser har antingen private eller protected konstruktörer.

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, klassen Sprite vi skapat uppfyller detta.

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

3.6. [**Ja**/Nej/Delvis] Det finns inte något minnesläckage, dvs. jag har testat och försökt se till att dynamiskt allokerat minne städas bort.
Kommentar: Ja, vid borttagande av objekt i spelet så städas de bort via vår Game Engine.

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 har stöd för både tangenter och musklick.

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, vi har använt oss av SDLs inbyggda is_intersecting metod och lagt till en wrapper för den i vår Game Engine.

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 och SDL_ttf, SDL_image.
Kommentar: Vi har olika Makefile kommandon för Windows, Linux och Mac. Vi har dock enbart testat att köra på Mac och Linux (btw) då vi inte har tillgång till en Windows-burk. Vi har inte medvetet använt några plattformsspecifika instruktioner och det bör därför fungera på Windows också.

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, båda spelen vi utvecklat har beteenden vid kollision och rör på sig på olika sätt.

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, båda spelen har minst två olika objekt och har flera instanser av minst ett av dem.

4.3. [**Ja**/Nej/Delvis] Figurerna kan röra sig över skärmen.
Kommentar: Ja, både med input och av sig själva.

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, figurerna flyttar på sig.

4.5. [**Ja**/Nej/Delvis] En spelare kan styra en figur, med tangentbordet eller med musen.
Kommentar: Ja, det går med tangentbordet i båda våra spel.

4.6. [**Ja**/Nej/Delvis] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.
Kommentar: Ja, det finns kollisionsbaserade event i spelen.