CPROG Rapport för Programmeringsprojektet

[Gruppnummer: 14]
[Gruppmedlemmar: Karl Rameld 9504261232]

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

Spelet är en Space Invaders kopia där spelarfigurer flyttas med hjälp av WASD och spelaren skjuter med musknappen. Som resursväg har ”./resources/” andvänts. Spelmotor i detta projekt motsvarar allt som ligger utanför main i princip. Grundtanken är att klasser som tillämpningsprogrammeraren vill lägga till skapas i main för att denne ska behöva göra så lite som möjligt, iteraktion mellan objekt och liknande funktionallitet läggs alltså ett lager ovanför tillämparen medan väldigt specifika funktioner som vad klasserna faktiskt gör och deras rörelse skrivs av tillämparen.

2. Instruktion för att bygga och testa

Koden som använts som grund är den från F13 på ilearn. Alla tidigare inlagda kommandon för mac eller linux som tillhandahållits mig finns kvar, så för att bygga ska förhoppningsvis bara makefilens kod behöva ändras. Spelaren kontrolleras som sagt med mus och WASD, det finns även ett kortkommando inlagt på spacebar. Spelaren får poäng(syns på texten i övre vänstra hörnet) genom att skjuta fiendeskepp, spelaren får minuspoäng om denne skjuter ett jordklot.

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, alla klasser tillämpningsprogrameraren använder sig av för att skapa sin implementations klasser ärver av en gemensam grundklass.

3.3. [Ja/Nej/Delvis] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.
Kommentar: Ja, den enda klassen som det går att skapa instanser av från början är Label, där konstruktorn är privat och tilldelningsoperatorer borttagna.

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

3.5. [Ja/Nej/Delvis] Inkapsling: datamedlemmar är privata, om inte ange skäl.
Kommentar:
Delvis, grundkomponentens SDL_Rect är protected. Detta då subklasser vart mycket enklare att ge mer komplexa rörelsemönster om SDL_Recten kunde manipuleras direkt av dessa. Jag vet att det finns lösningar runt detta men detta var i min mening det bästa sättet att koda spelmotorn på.

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, eller jag hoppas innerligt det.

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, gör i Session.cpp med hjälp av SDL_PollEvent för musen och SDL_GetKeyboardState för tangentbordet, detta sker i två separata loopar då jag ansåg spelarobjektet röra sig betydligt mycket mer responsivt med denna typen av lösning, t.ex. eftersom det enklare blir möjligt att separera rörelsehändelser från skjuthändelser och få dessa att hända samtidigt, det blir också lättare att få spelaren att röra sig diagonalt.

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, metoden collosion i 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 SDL 2 och SDL2_ttf, SDL2_image och SDL2_mixer.

Kommentar: Ja, se Instruktion för att bygga.

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

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

4.3. [Ja/Nej/Delvis] Figurerna kan röra sig över skärmen.
Kommentar: Ja, de figurer som bör kunna röra sig kan det.

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

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

4.6. [Ja/Nej/Delvis] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.
Kommentar: Ja, alla objekt interagerar dock inte med varandra.
Interaktionshantering görs på Components type-värde och det är möjligt att få alla typer av objekt att interagera med alla andra.