CPROG Rapport för Programmeringsprojektet

[Gruppnummer: 27]
[Gruppmedlemmar: Jennifer Skopac 000822, Patrik Svensson 960421

1. Beskrivning

Se ReadMe.txt filen

2. Instruktion för att bygga och testa

Se ReadMe.txt filen

3. Krav på den Generella Delen(Spelmotorn)

1. **[Ja/Nej/Delvis] Programmet kodas i C++ och grafikbiblioteket SDL2 används.**
   Kommentar: Ja, programmet är kodad i C++ och vi har använt SDL2 för grafiska komponenter.

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, vi har t.ex. basklassen Sprite som har flera subklasser och alla datamedlemmar är privata.

3. **[Ja/Nej/Delvis] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.**
   Kommentar: Ja, vi har tagit bort kopieringskonsturktorn och tilldelnings operatorn i klassen Sprite eftersom det är den enda polymorfa klassen vi har för tillfället.

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, vår gemensamma basklass är Sprite som också är en rotklass till de komponenter som är specifikt för vårt pacman spel. Exempel på sådana komponenter är Ghost och Dot.

5. **[Ja/Nej/Delvis] Inkapsling: datamedlemmar är privata, om inte ange skäl.**
   Kommentar: Ja, alla datamedlemmar i våra klasser är privata.

6. **[Ja/Nej/Delvis] Det finns inte något minnesläckage, dvs. jag har testat och sett till att dynamiskt allokerat minne städas bort.**
   Kommentar: Ja, vi har bl.a. använt shared_ptr smart pointers som hanterar objekt under deras livslängd. Vi städar också bort textures och frigör surfaces när de inte längre används och när spelet är klart så städar vi även bort renderer och SDL fönstret.

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 GameEngine's run metod har vi en game loop som tar emot olika event och sedan skickar det vidare till tillämpningens objekt. För att hålla spelmotorn modulär ansåg vi det lämpligt att låta metoden *tick* ta emot SDL eventet som en parameter. På så sätt kan varje specifik sprite hantera eventet och uppdatera sitt tillstånd direkt i sin egen *tick*-metod. Vi skulle

självklart kunna ha tangentinput i självaste gameloopen men då blir spelmotorn mer anpassad till ett specifikt typ av spel. I detta fall skulle det vara okej eftersom många spel tar emot tangent input för att gå fram och tillbaka, upp och ned osv. Men i det långa loppet kanske en tillämpningsprogrammerare vill bestämma själv vilka typ av input som påverkar sina sprites.

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. Detta genom en boolesk virituell metod i Sprite som kollar ifall två sprites har kolliderat med varandra. Sedan har vi även en annan virituell metod i Sprites subklasser som ändrar objektets tillstånd beroende på vilken sprite den kolliderar med. Denna metod beter sig annorlunda beroende på vilken subklass av sprite som överskuggar den. Exempelvis om Pacman kolliderar med en instans av Ghost så tas Pacman bort och spelet avslutas. Men om en Pacman kolliderar med en instans av Dot så försvinner Dot, eftersom Pacman “äter” den.

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, vi har inte använt några plattformsspecifika konstruktioner.

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

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. Spelaren styr pacman som kan åka över hela fönstret. Ghosts rör sig automatiskt uppåt och neråt och spelaren ska försöka undvika dessa, möter pacman och ghost varandra så försvinner pacman. Dots ligger över hela spelvärlden och spelaren ska äta upp dessa, när Pacman åker över en dot så försvinner dot.

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, i vårat spel finns en pacman, flera ghosts och flera dots

3. **[Ja/Nej/Delvis] Figurerna kan röra sig över skärmen.**
   Kommentar: Ja, spelaren kontrollerar pacman som kan röra sig över skärmen och ghosts rör sig automatiskt uppåt och neråt.

4. **[Ja/Nej/Delvis] Världen (spelplanen) är tillräckligt stor för att den som spelar skall uppleva att figurerna förflyttar sig i världen.**
   Kommentar: Ja, man kan se att figurerarna rör på sig.

5. **[Ja/Nej/Delvis] En spelare kan styra en figur, med tangentbordet eller med musen.**
   Kommentar: Ja, spelaren kan styra pacman med hjälp av tangentbordet. Pacman kan röra sig uppåt, neråt, åt vänster och åt höger inom spelvärlden.

6. **[Ja/Nej/Delvis] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.**
   Kommentar: Ja. För att simulera att Pacman äter Dots så försvinner varje dot som pacman

möter. Om Pacman möter en Ghost så försvinner istället Pacman och spelet är över.