LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Kodgranskning

Författare

Love Bäckman, lovba497@student.liu.se
Gustav P Svensson, gussv375@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2016
Version 1.0

2016-12-09

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
0.1	Utkast klart	161208
1.0	Renskrivning	161209

2 Möte

Innan mötet hade vi lagt upp vår kod på GitLab, vi hade dock en rätt rörig struktur så vi fick reda upp det en del och lägga upp det igen. Båda grupperna hade även skrivit varsitt kodgranskningsdokument (detta dokument) som vi under mötet lät den andra gruppen läsa igenom.

Mötet ägde rum den 9:de december 2016 klockan 13:00.

3 Granskat projekt

Spelet vi granskade och gav feedback på, "Arcaball-X", går ut på att spelaren ska studsa runt en boll och förstöra alla block på banan. Bollen studsas med hjälp av en plattform längst ned på banan som spelaren styr med musen.

3.1 Granskning

"Arcaball-X" är uppdelat i följande klasser:

- GameEngine
- GameLoop
- GameMenu
- GameState
- Graphic
- ResourceManager

I samtliga klasser borde gruppen överväga att använda "container.at(x)" istället för "[]-operatorn". Fördelen med "container.at(x)" är att ett error throwas om indexet inte finns, tillskillnad från "[]-operatorn" som ger upphov till odefinierat beteende i sådana fall. Detta är framförallt bra under utveckling, då det minskar risken för märkliga och svåridentifierade fel. I de fall en kontroll av vektorns storlek görs innan åtkomst är det dock ingen fara.

Konstruktorer borde även definieras i cc-filen istället för h-filen för samtliga klasser, för tillfället både deklareras och definieras konstruktorer i h-filerna.

3.1.1 GameEngine

Av vår förståelse så innehåller klassen GameEngine funktioner och variabler som hanterar hela spelet. Det är denna klassen som används från main filen, funktionen "startGame()" startas och den styr sedan hela spelet.

Variablerna “brickHeight” och “brickWidth” som finns i denna klassen tycker vi inte borde finnas här utan borde finnas i en brick klass istället då detta hör till brickorna. En stor fördel med detta är att man kan ha brickor med olika height och width vilket kan vara interessant på en bana.

De har även instanser av andra klasser sparade som medlemsvariabler i GameEngine, detta tycker vi är väldigt snyggt gjort då de kan komma åt exakt allt de behöver från denna klassen.

I funktionen “GameEngine::startGame()” så hittade vi två saker som kan förbättras, på rad 34 så finns det en while-loop utan några måsvingar, det blir väldigt mycket tydligare om man har while och if-satser med måsvingar, speciellt om det ligger en helt if-sats i while-loopen vilket det i detta fallet gör.

Allt detta ligger även i en while-loop som har “while(!done!)” som argument, det blir mycket tydligare om man istället har “while(window.isOpen()” som argument då man enklare fattar att denna loopen kommer köras sålänge spelet är på.

I destruktorn så deletar de även bara de första objektet av varje typ då de inte använder en for-loop. De borde loopa över varje typ och deleta den typen för att undvika minnesläckor.

3.1.2 GameLoop

Klassen GameLoop tycker vi är bra designad, det enda vi kunde hitta började på rad 87:

```
m_gameObjects[Balls][0]->upDate();
m_gameObjects[Pads][0]->move();
//draw all current objects
m_gameObjects[Balls][0]->move();
m_graphic->drawWindow(m_gameObjects[Balls][0]->draw());
m_graphic->drawWindow(m_gameObjects[Walls][0]->draw());
m_graphic->drawWindow(m_gameObjects[Walls][1]->draw());
m_graphic->drawWindow(m_gameObjects[Walls][2]->draw());
m_graphic->drawWindow(m_gameObjects[Pads][0]->draw());
```

Här målas inte alla gameObjects ut utan bara de första, för att lösa detta måste de implementera en for-loop som loopar igenom alla gameObjects.

3.1.3 GameMenu

Klassen GameMenu ärvs av klassen GameState, vilket är snyggt då de kan återanvände kod från den klassen. Det enda vi hittade var hårdkodade värden för namn och poäng på rad 146:

```
std::string name{"Christer"};
std::string point{"12334"};
```

3.1.4 GameState

GameState är bas-klassen för GameMenu, det enda den innehåller är en virtual funktion som returnerar en enum CurrentState.
Vi tycker inte att konstruktorna ska vara protected utan den borde vara public då det är standard i C++.
Vi tycker även det är tydligare att skriva:

```
GameState() = default;
```

istället för:

```
GameState() {}
```

3.1.5 Graphic

Av vår förståelse så hanterar Graphic klassen spelfönstret. Vi har inga kommentarer om denna klassen då vi tygger att designen av denna klassen är bra.

3.1.6 ResourceManager

Precis som i GameState så tycker vi att konstruktorn och destruktorn ska vara default om man bara har tomma måsvingar efter den då det blir tydligare vad de gör.

Vi tycker namnet är lite otydligt då när vi bara läste klassnamnet så lät det som att klassen hanterar pekare och andra resurser, ett bättre namn skulle kunna vara "loader" då det enda den gör är att ladda fonts och bilder.

3.2 Använding av const

Vi hittade flera ställen där de bör använda const på funktioner som inte ändrar några värden.

Följande funktioner borde vara const då de inte ändrar på några värden:

- Funktionen: "GameMenu::seHighScore()"
- Funktionen: "Graphic::isWindowOpen()"
- Funktionen: "Graphic::getEvent()"
- Funktionen: "ResourceManager::getTexture"
- Funktionen: "ResourceManger::getFont"

Följande parametrar borde vara const-referenser för att undvika kostsamma kopior:

- Funktionen: "Graphic::setUpWindow()", parametern "std::string name" borde vara en const-referens.

Följande parametrar borde vara const-referenser då de inte modifieras:

- Funktionen: "GameLoop::GameLoop()", parametern "ent" är en referens men borde vara en const-referens.

3.3 Namngivning

Följande funktionsnamn och/eller variabelnamn tycker vi kan ändras för att bli tydligare:

- Funktionen “Graphic::setUpWindow()” borde heta “Graphic::setupWindow()”. Det finns t.ex “setupText()” och “setupMarkerVectors()” funktioner som följer den senare namnkonventionen.
- Funktionen: “GameEngine::setUpGraphic()” borde heta “GameEngine:setupGraphic()”.
- Funktionen: “GameMenu::seeHighScore” borde heta “GameMenu::seeHighscore()” eftersom Highscore är ett ord.

3.4 Minneshantering

Som vi skrev innan så finns det minnesläckor i koden just nu. I destruktorn för klassen “GameEngine” så tas inte alla “gameObjects” bort, bara den första av varje typ vilket leder till minnesläckor.

3.5 Vårt projekt

Vi fick bra feedback på vår kod, gruppen som granskade vår kod tyckte att vår filuppdelning var väldigt förvirande då vi har flera olika mappar med olika verisioner av spelet. När vi väl visade vilken mapp de skulle kolla i gick det dock lite bättre.

De hade dock lite problem med att förstå vår kod då själva spelet inte kompilerade och alla filer inte var skapade, vilket var en bieffekt av att vi var mitt uppe i arbetet att omstrukturera vår kodbas.

Ang. klasstrukturen för våra objekt håller vi inte med i kritiken då vi anser att vår struktur gör det väldigt enkelt att lägga till nya objekt med andra egenskaper än tidigare objelt.