TDP005 Projekt: Ojektorienterat system

Kod-granskningsprotokoll

Författare

Pontus Haglund, ponha098@student.liu.se
Niklas Nilsson, nikni292@student.liu.se

Höstterminen 2015
Version 0.1

2015-12-10

Innehåll

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
0.1	Dokumentet upprättat	1512110

2 Inledning

Följande dokument kommer redogöra för det kod-granskningsmöte som ägde rum den 9e december 2015. Redogörelsen kommer vara strukturerad i tre huvudrubriker som följer nedan. Först kommer detta dokument redogöra för mötet i sig, sedan för kritiken som vi gav de andra grupperna och sist kritiken vi tog emot. Under den sista rubriken kommer också verkan av denna kritik diskuteras.

3 Mötet

Kod-granskningsmötet ägde rum den 9e december 2015. Efter en kort promenad runt i b-huset för att hitta en bra plats att genomföra mötet på retirerade vi tillbaka till grottan (den större IP-salen). Anledningen till reträtten grundar sig i att vi inte kunde hitta någon annan plats att vara. Väl där genomförde vi ett ganska informellt och kort möte där vi i ingen särskild ordning framförde kritik till andra grupper.

Mötet planerades den 8e december 2015. När detta planerades nåddes en överenskommelse att alla grupper skulle sätta sig in i de andra gruppernas kod. Vi hade skrivit ner kommentarer på anteckningspapper och gav varandra kritik utifrån dessa anteckningar och antecknade den kritik vi fick. Mötet var ganska allmänt i utformningen och det var sällsynt att specifik kod togs upp utan det var mer generella kommentarer, med några undantag.

Vi ville gärna bli av med detta möte och granskningen så fort som möjligt eftersom vi känner ganska stor press över att få klart projektet. Således kanske detta möte genomfördes hastigare än det borde och inte planerades i den utsträckning som var möjligt. Ett exempel på annan utformning som kunde varit konstruktiv kan ha varit att i detalj granskat exempel från olika gruppers kod och tillsammans pratat om hur man kunnat löst det bättre.

4 Kritik till andra grupper

Vår kritik till de andra grupperna kommer här delas upp i två underrubriker. Det vi under denna rubrik tar upp om gruppernas kod är inte på något sätt allt vi nämde om gruppernas kod utan snarare ett urval av de punkterna som vi la mest vikt vid.

4.1 Space shooter - Filip Johansson och Viktor Sandgren

Kritiken som vi gav till Filip och Viktor var övervägande mycket positiv. De hade lyckats göra saker som vi drog lärdomar av. En detalj vi tog upp tidigt i mötet var att deras design av klasser verkade bra. Vidare tyckte vi om att deras spelobjektet själva stoppade in sig i en vektor när de initierades. Nedan är ett exempel från deras konstruktor av fiende objekt:

```
Enemy::Enemy(int hit_points, double x_velocity, double y_velocity, string image, int start_x,
    int start_y, SDL_Renderer* renderer, int sc, vector<Sprite*> &update_list)
: Sprite(hit_points, x_velocity, y_velocity, image, start_x, start_y, renderer), score{sc}
{
        update_list.push_back(this);
}
```

Redan mängden argument som skickas till denna konstruktor och som annars hade skickats runt individuellt mellan funktioner berättigar skapandet av klassen ovan [1, p. 152-153]. Men vi tyckte just Enemy var ett bra exempel på andra saker McConnell tar upp rörande klass-design. Nedan följer header-filen till klassen Enemy:

```
class Enemy : public Sprite {
public:
        Enemy(int hit_points, double x_velocity, double y_velocity, string image, int start_x,
            int start_y, SDL_Renderer* renderer, int sc, vector<Sprite*> &update_list);
        ~Enemy();
//      bool collision_check() override;
        void movement(double time_interval);
        int get_score() const;

private:
        int score;
};
```

Även om denna klass inte är implementerad till fullo kan vi se exempel på det McConnell [1, p. 152-154] kallar för att gömma komplexitet så att programmet blir lättare att förstå. Något som vi påpekade för Filip och Viktor. De döljer här flera delar av sitt program som hade ökat programmets komplexitet. Ett exempel som vi redan ovan berört i korthet är volymen av argument nog för att visa på detta. Men vi kan i header-filen se fler exempel där objektet själv kan (eller kommer att kunna) lösa förflyttning och kollision själv. Vi kan även se exempel på inkapsling i denna klass [1, p. 138-141]. Vi kan till exempel se att score inte kan hämtas direkt utifrån utan det finns en passande getter.

Något vi kritiserade i deras projekt var att de skapat en praktisk window klass som bland annat hanterade SDL_Window och SDL_Renderer. Konstigt nog användes inte den.

```
        Level lvl1{1, wave};
        lvl1.run(ren, update_list, window_w, window_h);
```

Ovan är ett exempel där Filip och Viktor skickar 3 argument till ett Level objekt istället för att använda sig av denna praktiska klass de skapat. De kunde antingen använt getters för att skicka dessa argument eller skickat med en konstant referens till objektet och packat upp detta inuti lvl1.

För att hålla ner längden av detta dokument kommer vi inte ge fler detaljerade exempel från Filip och Viktors kod. Några saker vi inte tagit upp i detalj här men som framfördes till gruppen är:

En diskussion om för och nackdelar till att de bara hade en nivå av objekt vi granskningstillfället.

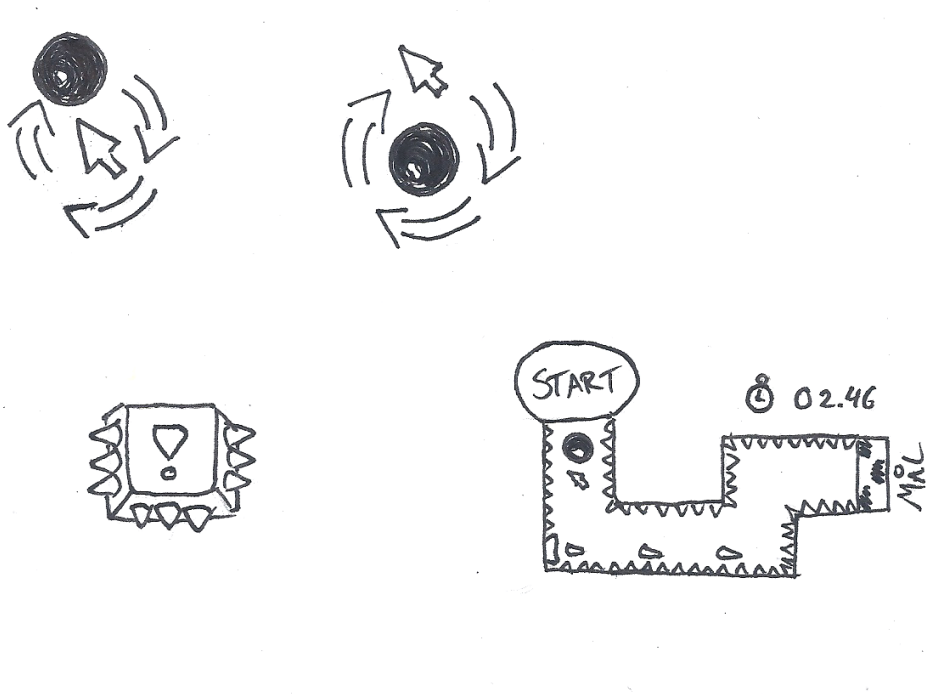
Fler kommentarer skulle behövas för att göra koden tydlig och läsbar för utomstående.

Bättre uppdelning av vissa långa funktioner.

Mer och utförlig dokumentation av projektets struktur.

4.2 Orbit alpha - Emelie Olmås och Eric Forsberg

Det första vi tog upp rörande Emelie och Erics projektet var att deras dokumentation gav en mycket tydlig bild om målet med deras projektet innan vi ens tittat på koden eller kompilerat något. Se nedan:



Figur 1: Prototypritning av Eric och Emelie

Vi påpekade också att denna grupp lyckats bättre med kommentarer än någon annan av oss. De tillhandahöll också en Design notes.txt som innehöll en tydlig om något ut-daterad fil där varje klass kortfattat beskrevs. En kritik vi framförde i relation till detta var att vissa delar av koden var onödigt kommenterad där det var uppenbart vad koden gjorde. Nedan följer ett exempel där det är fullkomligt uppenbart vad koden har för uppgift:

```
void Map::draw(SDL_Renderer* renderer)
{
        //Render Spike clusters.
        for (Spike* spike : spikes)
        {
                spike->draw(renderer);
        }
```

Vi framförde även kritik rörande bristande inkapsling på en plats i deras kod. Detta ledde till en ganska intressant konversation rörande vad poängen med att kapsla in denna offentliga variabel var.

```
...
        level.bubble->set_resting(false);
...
```

Ovan är alltså ett exempel där den ofentliga variabeln hämtas ut med hjälp av objektet i spel-loopen. Nedan följer ett utdrag ur klassen i fråga.

```
class Bubble : public Orbital
{
        private:
                ...

        public:
                Bubble(SDL_Renderer* renderer, std::string filename, int x=0, int y=0);
        ...
                void set_resting(bool state);
                ...
```

```
                int get_x_resting() const;
                ...
};
```

Diskussionen vi hade rörande detta berörde punkerna McConnell [1, p. 152-154] tar upp i Code Complete. Några intressanta punkter från diskussionen var fördelar och nackdelar med att kapsla in medlemmarna och att det redan fanns inkapsling i klassen. I exemplet ovan kan vi till exempel se en getter och en setter för privata medlemmar.

5 Kritik från andra grupper

Kritiken som vi fick under detta möte var mycket lik den vi gav. De som deltog på mötet tyckte i princip att vi alla lyckats med att producera klasser som var tydliga och designade med ett tydligt syfte.

Vi fick dock ganska gott om tips som kommer att och redan har ändrat hur vår kod ser ut och fungerar. Det första sättet var att vi såg hur de andra grupperna löst problemet med att abstrahera bort det mesta ur Main programmet. Detta var något vi innan kodgranskningen fortfarande inte riktigt visste hur det skulle gå till. Tack vare detta granskningstillfälle och grupperna som deltog har vi bara en dag senare tre mer eller mindre fungerande klasser som sköter huvudmeny, spel och menyn efter spelet är slut.

Vi fick också bra kritik rörande de includes som vi gör i början av de flesta header och ibland även implementationsfiler. Dessa includes tar ibland upp över 20 rader och är något vi bör åtgärda. Detta är något som började med en felaktig ordning av includes i main programmet och som sedan smittat våra olika klasser. Det är inte ett problem från datorns perspektiv så klart eftersom det finns guards över allt. Men det är knappast någon fördel att alla filerna är längre och att det är svårare att se vilka saker varje klass faktiskt använder sig av.

Kritik rörande variabel-namn framfördes också till oss. Framförallt i argumentlistan i våra konstruktorer så bröt vi mot många av de regler och riktlinjer som McConnell [1, p. 259-261] tar upp. De flesta variablerna som togs emot av konstruktorn fick endast en bokstav som namn. Tanken bakom dessa variabel-namn är att det var praktiskt att skriva dem samt att det blir uppenbart vad de står för när de initieras. Men som McConnell tar upp [1, p. 267-270] är det bra att ge tydliga namn även till variabler som är temporära. Vissa av våra variabler skickas även iväg till andra konstruktorer vilket gör det svårt för en läsare att följa koden.

Referenser

[1] Steve McConnell, *Code Complete*, Microsoft Press, USA, 2nd edition, 2004.