LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Kodgranskning

Författare

David Dumminsek, davdu153@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2022
Version 0.1

2022-12-07

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
0.1	Första utkast	150603

2 Mötet

Vi planerade att ha vårt möte den 6 december runt klockan 10. Vi hade dock inte planerat något att göra inför mötet så båda grupper var oförberedda. Under mötets gång så bestämde vi oss för att dela varandras gitlab projekt för att kunna närmare granska koden i efterhand och om eventuella frågor skulle komma upp så kan de ställas på discord.

Under mötet så gick vi igenom varandras kod och förklarade vad de gjorde. Samt gav vi kritik till varandra om vi hittade något men var svårt att veta för att vi inte kollade på något att koden i förväg.

3 Granskning

Spelet som jag granskade var i en väldigt tidig stadie. Än så länge verkar det bara vara en startmeny. Det eventuella spelet ska vara ett towerdefense spel.

3.1 Kommentarer

Det fanns inga kommentarer. Det skulle underlätta en hel del och göra det lättare att förstå vad koden gör i vissa delar. Läsbarheten av koden är dock inte så bristande men den skulle ändå kunnat ha förbättras med lite kommentarer.

3.2 Engine

Den klassen som hanterar game loopet kallas **Engine**. De använder **sf::RenderWindow::setFramerate-Limit** funktionen vid slutet av varje loop. Det känns onödigt då det bara behöver sättas en gång i början.

3.3 Game States

De övriga klasserna som de använder är klasser för så kallade game states”. De är uppbyggda av en bas klass som heter **Base_state** och 3 subklasser vid namnen **Menu_State**, **Game_State** och **Game_Over_State**.

Tanken bakom dessa är att de ska olika render och update funktioner som gör olika saker. Till exempel **Menu_State** klassens render funktion ska rendera en meny.

I **Base_State** header filen så har de deklarerat alla game state klasser i samma fil. Det vill man helst undvika då man vill ha en header fil och en motsvarande cc fil för varje klass. Det blir mycket lättare för programmeraren att leta efter en header fil istället för att skrolla igenom en header fil för att hitta en klass deklaration.

Base_State klassen har inga datamedlemmar men alla subklasser har en datamedlem med samma namn **curr_state**. Eftersom den används av alla subklasser ser jag ingen anledning till varför den inte kan vara en datamedlem i **Base_State**.

3.4 Användning av const

De tar ingen användning av **const** när det kommer till funktioner. Det är bra praxis att använda **const** då det är som att göra ett kontrakt med kompilatorn som berättar att den här funktionen inte kommer ändra på objektet. Så om man av misstag lägger till något i en **const** funktion som ändrar på objekten så kommer den inte kompilera.

3.5 Minneshantering

De har inget som kräver minneshantering ännu (använder inte new eller malloc).

4 Mitt projekt

Deras kommentarer om mitt projekt Dogeater.

4.1 Kommentarer

Min kod har också en brist på bra kommentarer då jag enbart fokuserat på att få något som fungerar. Men det bör inte vara några svårigheter att lägga till.

4.2 Main

Min main fil innehåller 25 rader kod. De tycker att jag borde sträva att få den lik exemplet på kurshemsidan. Det skulle jag enkelt kunna fixa med hantera game loopet i en annan klass lik deras **Engine** klass.

4.3 Game

Min **Game** klass är väldigt stor och har väldigt många ansvar. Klassen har alltså väldigt låg ”cohesiondå jag bör sträva efter hög ”cohesiondär varje klass och funktion har ett specifik uppgift.

Det skulle jag kunna fixa med att dela upp **Game** klassen i många olika mindre klasser. Till exemepel kan jag ha en **objectManager** klass som uppdaterar och initierar varje objekt.

4.4 default deklarering

Jag använder mig av default deklaring till alla mina **Entity** move- och copyassignment funktioner. Det är något jag borde starkt undvika påstår de. Anledningen till varför jag använder av default är att jag inte är helt klar med mina **Entity** klasser och har inte fått några problem än så länge (det går att kompilera).

Men det kommer eventuellt fixas när jag har definierat klart alla klasser.

4.5 Användning av const

Jag har också väldigt dålig använding av **const**.

Det kan enkelt gå att fixa igenom att kolla igeom alla funktioner och applicera **const** där det passar.

5 Övriga brister

Några brister jag kom på eller hittade gällande mitt projekt.

5.1 Rendering av fiende projektiler

Jag märkte ganska snabbt att mitt spel börja lagga när jag behövde rendera många fiende projektiler. Jag kom fram till att det var min for loop som använder **sf::RenderWindow::draw** på varje **sf::Sprite**. Eftersom det blir en del fiende projektiler att rita och jag kallar draw funktionen väldigt många gånger så kan det blir väldigt laggigt.

För att lösa det måste jag reducera hur många gånger jag kallar draw funktionen. En populär lösning på problemet är att använda **sf::VertexArray** då behöver jag bara göra en kallelse på draw funktionen för varje **sf::VertexArray**, istället för en kallelse för varje element i en vektor av sprites.