LINKÖPINGS UNIVERSITET

TDP005 Projekt: Objektorienterat system

Kodgranskningsprotokoll

Författare

Lukas Freyland, lukfr510@student.liu.se
Elliot Johansson, elljo130@student.liu.se
Nadim Lakrouz, nadla777@student.liu.se

Höstterminen 2022
Version 1.0

2022-12-07

INSTITUTIONEN FÖR DATAVETENSKAP

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.0	Första versionen av protokollet	221207

2 Möte

Mötet var den 6 december kl. 10.30. Vi gick igenom varandras kod och redovisade vad vi hade gjort fram tills mötet samt hur vi låg till. Under mötet diskuterade vi bristerna med våra projekt och hur projektet kunde förbättras samt vilka styrkor båda projekten hade. I slutet av mötet delade vi våra gitlabprojekt så vi kan titta igenom koden senare samt för att jämföra koden med designspecifikationen.

3 Granskat projekt

Spelet vi granskade kalladas för Dogeater. Dogeater är ett ”bullet hell” spel vilket betyder att spelet går ut på att skjuta ner inkommande fiender med projektiler samtidigt som spelaren försöker undvika en stor mängd av projektiler som fiender har skjutit. På spelplanen finns en spelare i form av ett flygplan samt fiender som kommer fram när spelplanen har skrollat upp till dem.

3.1 Gameloop

Gameloopen i det granskade projektet är för stor, enligt kurshemsida[1] så ska loopen vara så liten som möjligt. Det kan lösas genom att dela upp koden i två klasser, ’Engine” och ”Main”. Där ”Engine” hanterar alla klasser och funktioner i spelet och ”Main” hanterar körningen av programmet.

3.2 Game

Klassen *Game* i spelet är för stor.

Game hanterar mer än vad som behövs i den klassen vilket den klassen inte borde. Funktionerna mellan klasserna borde delas upp mer. Till exempel så borde inte *Game* hantera kollision. ”CollisionCheck” kan hanteras internt i *Player* klassen eller *Enemy* klassen. ”SpawnEnemy” Funktionen kan ligga i *Enemy* manager som kan också hantera ”update” och ”render”. Hanteringen av JSON-filen sker också i *Game* klassen som borde hanteras i en dedikerad klass. Det i sin tur skulle förbättre objektorienteringen av klasserna.

[1] https://www.ida.liu.se/∼TDP005/current/

3.3 Minnesläckor

Det fanns ej något konkret sätt att hantera minnesläckor i koden. Det skapar stora problem då speler kommer börja lagga och till slut krasha, då minnet tar slut.

3.4 CMakeFil

Dogeatergruppen har lagt flaggorna i en alias i bashrc-filen istället för att förvara dem i CMake där man lättare kan komma åt dem och ändra om det skulle behövas.

3.5 Dokumentation av projektet

Koden för Dogeater spelet saknar kommentarer vilket gör koden mindre läslig och det går inte att generera en referensmanual med Doxygen isåfall. Enligt kurshemsida[2] ska koden av projektet kommenteras enligt Doxygen för att kunna skapa en meningsfull dokumentation.

3.6 Positiv feedback

Kommentarer om vad som var bra med projektet Dogeater.

3.6.1 JSON hantering

Data hantering med JSON-fil var bra. Det gav mycket flexibilitet till projektet då det blir lättare att skapa flera olika scenarion för samma spel. Och det är också bra för att då behövs ingen kodändring för att skapa de olika scenarion utan den ändringen sker i JSON-filen.

3.6.2 Pekare

Dogeatergruppen använde pekare flera gånger och på ett flitigt sätt.

4 Vårt Projket

Kommentarer på vårt projekt.

4.1 Const

Dogeatergruppen påpekade att vi kunde öka användningen av ”const” bland annat i get-funktioner. Det går lätt att fixa då vi behäver bara lägga till const till de relevanta medlemsfunktionerna

[2] https://www.ida.liu.se/∼TDP005/current/

4.2 Uppdelning

Vi hade för mycket uppdelning bland klasserna.

4.3 main

Main klassen hade lite innehåll. Men det ska den vara.