Eksamen for Software Design

Prosessen

Vi startet med å diskutere hvordan vi skulle dele opp og gjennomføre eksamen. Her satt vi oss inn i hovedtemaene for eksamen, og skrev de ned i en samlet oversiktlig liste. Vi gikk gjennom hvert punkt og snakket om vi skulle implementere det i koden eller ikke. Hvis ja, også om hvordan vi skulle implementere det og hvor vi kunne bruke det for å vise forståelse i faget.
Design Patterns er en stor del av faget, og vi gikk gjennom hvilke patterns som ville være gunstige for oss å dra nytte av. Vi startet med å lage et Use Case Diagram da dette ville gi oss en rask og enkel oversikt over hvordan oppgavene i spillet skulle utføres og hvem som skulle utføre de. Mot slutten av kodingsprosessen lagde vi et sekvensdiagram for å se hvordan og hvilke elementer som snakket med hverandre. Dette gjorde vi fordi vi da lett kunne se hvilke klasser og interface som blir kalt på ofte. Løsningen på disse vil da være å dele opp i mindre klasser og Interfaces, så vi oppfyller prinsippene for SOLID. Etter dette lagde vi klassediagrammer for de viktigste klassene med interfacene sine også med.
Vi valgte å ikke bruke Domain model diagram. Dette selv om det hadde gitt oss en bra konseptmodell. Grunnen til dette var at vi ikke var erfarne nok med C# til å lage et godt nok Domain Model Diagram da dette krever kunnskap om selve språket. Vi gikk heller veien ved å kun gå ut ifra en enkel Use case modell og bygge oss opp fra en liten løsning som fungerer og bygge videre på den.
Sluttprosessen var å rydde opp i koden og dele den inn i egne mapper. Her har vi Cards,Dealers og Players.Vi har sørget for at koden er godt strukturert og lett å forstå for en utenfra. Den har gode klasse-, variabel-,metode- og interface navn, som beskriver innholdet på en tydelig måte. I sluttfasen ryddet vi også opp i dokumentet som beskriver prosessen av arbeidet. Siden vi gradvis fylte på men informasjon, var mye allerede gjort.

Parprogrammering

Vi benyttet parprogrammering på deler av løsningen, det førte til at vi kunne hjelpe hverandre på problemer som dukket opp og komme til en felles enighet i koden om hvordan vi skulle gå fram med ting. Verktøyene vi brukte til dette er discord, git og github med et privat repo for å hindre at andre skal fuske. Parprogrammering er et av verktøyene vi har brukt mye dette semesteret både i Avansert Java men også i Software Design, det både gode og dårlige ting med dette. Noen av de gode punktene er at man lærer av andre med å se på hvordan de koder og hvordan de løser problemet, men man lærer også mye av å lære bort det du kan for å få en mer grundigere forståelse av temaet som det dreier seg om, man får også snakket om flere løsningsmetoder og valgt den som passer best for denne løsningen. men det er også noen negative sider med dette hvis man ikke er flink å jobbe med andre. det kan være problem hvis en bare kjører over andre og bare jobber på selv om den andre ikke kan det like godt, om man sitter fast og blir heller kritisert hvorfor du ikke kan løse problemet en å heller få tips til hvordan og løse det. dårlig kommunikasjon er også en av problemene som kan dukke opp, hvis en har ikke hørt om et tema og han andre bare snakker i avanserte terminologi som han første ikke forstår.

Features fra pensum

1. Use Case Modell
2. Sekvensdiagram
3. Classe diagram
4. Factory
5. Singleton
6. Multithreading
7. Abstract klasser
8. SOLID

Spesielle utfordringer & fancy features

Vi viser hendene til spillerne etter vær gang det gjør en runde.
Håndtering av at ingen spesialkort kan trekkes før spillet begynner ordentlig, og at bomben og Gribben leverer et kort som ikke er spesialkort.
Vi håndterer errorer ved brukerens input og ber brukeren skrive et nytt tall innenfor parametrene som er gitt (2-4), gir en vennlig error melding om det skjer noe galt med dette.
Vi bruker både Singleton til å få alle spillerne skal spille mot samme bunke for å hindre at de spiller mot forskjellige Dealere.
Vi bruker Factory pattern for å få Konstruering på et sted der man trenger bare og bytte på det ett sted.
Vi har implementert en helt egen klasse for å håndtere de fleste Console.WriteLine. Denne klassen har navnet StandardMessage.cs siden mesteparten av det som blir skrevet ut i terminalen blir håndtert her.
Håndtering med lock og if statement til å hindre at noen andre skal kunne spille samtidig fordeles være thread safe, men også hindre at vi får deadlock eller en race condition.

Bugs

Har ingen kjerne bugs som vi har funnet.

Kilder

forelesnings slidene i PG 3302
https://plantuml.com/